intel®

APPLICATION NOTE

AP-236

September 1985

Implementing StarLAN with the Intel 82588 Controller

SHARAD GANDHI
SENIOR APPLICATIONS ENGINEER
DATACOMMUNICATION COMPONENTS OPERATION

Order Number: 231422-001

7.1 INTRODUCTION

The explosive growth of the personal computer market has placed a PC on almost everyone's desk in the office. Working in a stand-alone PC environment for a while automatically generates a need for linking up the PCs for very basic reasons, like file sharing, automatic back-up, disk-less operation and electronic mail. This has led to a search for a networking scheme for PCs which costs not more than 10% of the cost of the PC itself. As of year end 1984, close to 4.5 million PCs were in existence. Only 5% of these had local area network (LAN) interfaces. Industry forecasts suggest that by 1990, nearly 20 million PCs will be interconnected through LANs. To what extent this comes to pass depends on achieving low cost networking, ease of design, ease of installation and achievement of industry standards to enable inter-connectability of equipment from different vendors. Traditional Local Area Networks (LAN) like Ethernet and Cheapernet have proved to be too expensive for the office environment. Not only the nodes are expensive, but also the cabling. A myriad of non-traditional networks have emerged which are cheap. But none have captured a significant market, due to the lack of major company backing and also due to lack of backing from any of the standards bodies like the IEEE, CCITT or ISO.

Two recently introduced LANs will have a far reaching impact on PC networking. Based on the CSMA/CD (Carrier Sense Multiple Access with Collision Detection) access method, they are both targets of industry standardization efforts through the auspices of the IEEE 802.3 Working Group: a) PC Network introduced by IBM and Sytek for 2 Mb/s in broadband over a coaxial cable and b) StarLAN introduced by AT & T with a data rate of 1 Mb/s in baseband over unshielded, twisted pair, telephone grade wiring. The INTEL 82588 LAN controller was designed to support both types of networks equally well, being optimized for operation in the 1–2 Mb/s range, with either baseband or broadband transmission. It is a VLSI device aimed at low cost, ease of design, and conformance to the anticipated IEEE 802.3 standards.

The objective of this Application Note is to illustrate designing with the 82588 through a practical example. StarLAN was chosen for this purpose due to its overall simplicity. The Application Note goes beyond the 82588 based StarLAN interface, describing overall operation of the network, and providing an example design of a StarLAN HUB unit.

7.1.1 StarLAN

StarLAN overcomes the handicaps of cost and standards. It is very economical to implement; both, due to the low cost of the node and that of the cable. The cost of a node is drastically reduced because of the availability of VLSI LAN controllers like the 82588. StarLAN uses standard telephone wire for the transmission medium. This cable is either already installed and ready to use or it is very cheap to install. StarLAN has the backing of most of the major companies in the industry. And, in addition, since it meets the IEEE 802.3 standards requirements, it is very fast on its way to become an industry standard.

7.1.2 Network Topologies

Networks connect nodes so that they communicate. There are various ways of interconnection or topologies. Figure 7-1 shows the three most commonly used topologies; bus, ring and star.

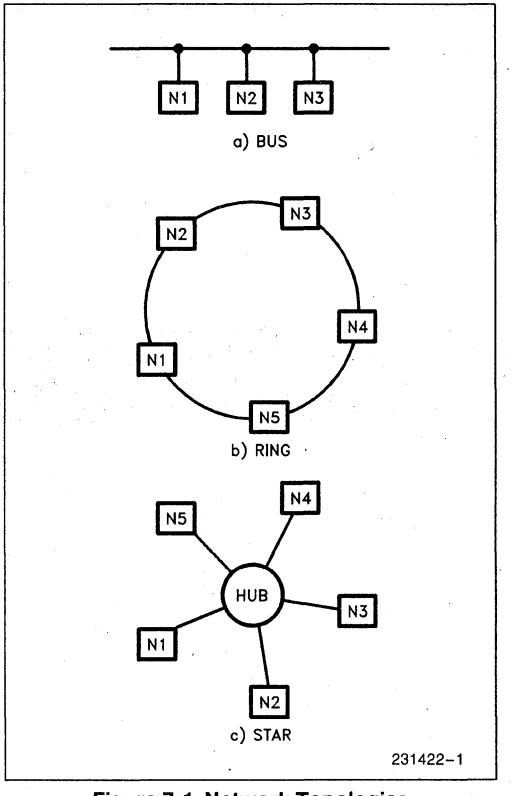


Figure 7-1. Network Topologies

In the bus topology, all the nodes are connected to the same bus or the transmission medium. There are various protocols to determine who can transmit. In the CSMA/CD (Carrier Sense Multiple Access with Collision Detect) protocol, any node can transmit if there is silence on the bus for a given time. If two nodes transmit at the same time, they collide. They detect the collision, wait for a random period of time and try transmitting again. Ethernet is an example of a LAN with a bus topology using the CSMA/CD protocol. No single node is critical to the network. Each is a peer.

If nodes are connected together to form a ring, there is generally a token which gets passed from node to node. Whoever has the token can transmit and then pass the token to the next node. Token ring network is an example of this topology. Although, Token bus network physically has a bus topology, the nodes form a virtual ring around which a token is passed. In a star topology, as in StarLAN, the nodes are connected to a central unit called the HUB. The HUB receives and retransmits the frames from each node. It is like a switching station in a telephone network. Since the HUB sits at a central place, it can perform functions which are shared by all nodes. StarLAN also uses a CSMA/CD protocol like the Ethernet. The HUB in the StarLAN network primarily aids in resolving collision among the nodes.

7.1.3 The 82588

The 82588 is a single chip LAN controller designed for CSMA/CD networks. It integrates in one chip all functions needed for such networks. Besides doing the standard CSMA/CD functions like framing, deferring, backing off on collisions, transmitting and receiving frames, it performs encoding and decoding the data in Manchester or NRZI format, carrier sensing and collision detection up to a speed of 2 Mb/s. These functions make it an optimum controller for a StarLAN node. It has a very conventional microcomputer bus interface, further easing the job of interfacing it to any processor.

7.1.4 Organization of the Application Note

Section 7.2 of this Application Note describes the StarLAN network, its basic components, collision detection, signal propagation and network parameters. Sections 7.3 and 7.4 describe the 82588 LAN controller and its role in the StarLAN network. Section 7.5 goes into the details of designing a StarLAN node for the IBM PC. Section 7.6 describes the design of the HUB. Both these designs have been implemented and operated in an actual StarLAN environment. Section 7.7 documents the software used to drive the 82588. It gives the actual procedures used to do operations like, configure, transmit and receive frames. It also shows how to use the DMA controller and interrupt controller in the IBM PC and goes into the details of doing I/O on the PC using DOS calls. Appendix A shows oscilloscope traces of the signals at various points in the network. Appendix B deals with doing DMA in environments where only one DMA channel is available.

7.1.5 References

For additional information on the 82588, see the LAN Components User's Manual or the 82588 Reference Manual. For additional information on StarLAN, see a draft of IEEE 802.3 type 1BASE5 specifications.

7.1.6 Acknowledgements

Intel Corporation gratefully recognizes AT & T Information Systems for the StarLAN concept and their contributions to the IEEE 802.3 1BASE5 Task Force.

Ideas and cooperation from Bob Galin, Adi Golbert, Ariel Hendel, Yosi Mazor and Kiyoshi Nishide have been very helpful.

7.2 StarLAN

StarLAN is a low cost networking solution aimed at the office automation, instrumentation and serial backplane applications. It is a 1 Mb/s, IEEE 802.3 compatible CSMA/CD network. It has a star topology with the nodes connected in a point-to-point fashion to a central HUB. HUBs can be connected in a hierarchical fashion. Up to 5 levels of HUBs are supported. The maximum distance between a node to the adjacent HUB or between two adjacent HUBs is 800 ft. (250 meters) for 24 gauge wire and 600 ft. (200 meters) for 26 gauge wire. Maximum node to node distance with one HUB is 0.5 km, hence IEEE 802.3 calls it a 1BASE5 LAN. 1 stands for 1 Mb/s and BASE is for baseband.

One of the attractive features of StarLAN is that it uses telephone grade twisted pair wire for the transmission medium. In fact, existing, installed telephone wiring can also be used for StarLAN. Telephone wiring is probably the cheapest wire. It is also very economical to install. Although use of telephone wiring is an obvious advantage, for small clusters of nodes the entire wiring can be done without using building wiring.

Factors contributing to its low cost are:

a. Use of telephone grade, unshielded, 24 or 26 gauge twisted pair wire transmission media.

b. Installed base of redundant telephone wiring in most buildings. Even new installation of telephone wiring is very economical.

c. Buildings are designed for star topology wiring. They have conduits leading to a central location.

d. Availability of low cost VLSI LAN controllers like the 82588 for low cost applications and the 82586 for high performance applications.

e. Low cost RS-422 drivers/receivers needed for the physical level interface.

7.2.1 StarLAN Topology

StarLAN has (as the name suggests) a star topology. The nodes are at the ends of the arms of a star and the central point is called a HUB. There can be more than one HUB in a network. The HUBs are connected in a hierarchical fashion resembling an inverted tree, as shown in Figure 7-2, where nodes are shown as PCs. the HUB at the base (at level 3) of the tree is called the Header Hub (HHUB) and others are called Intermediate HUBs (IHUB). It will become apparent, later in this section, that topologically, this entire network of nodes and HUBs is equivalent to one where all the nodes are connected to a single HUB.

7.2.1.1 TELEPHONE NETWORK

StarLAN is structured to run parallel to the telephone network in a building. The telephone network has, in fact, exactly the same star topology as StarLAN. Let us now examine how the telephone system is laid out in a

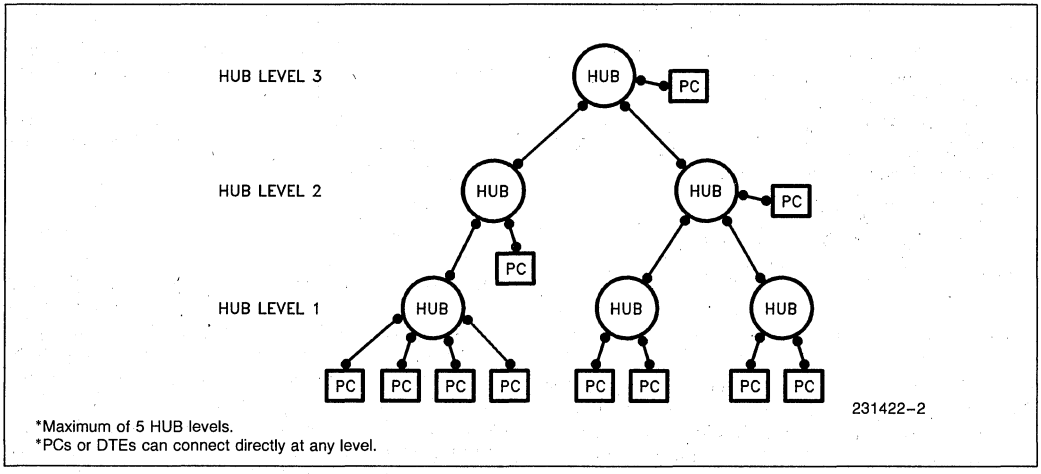


Figure 7-2. StarLAN Topology

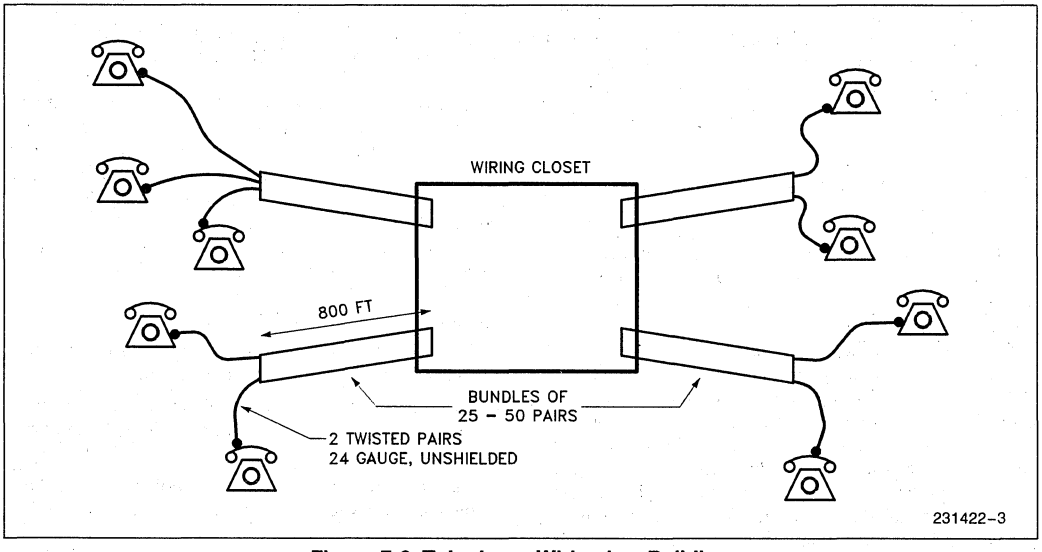
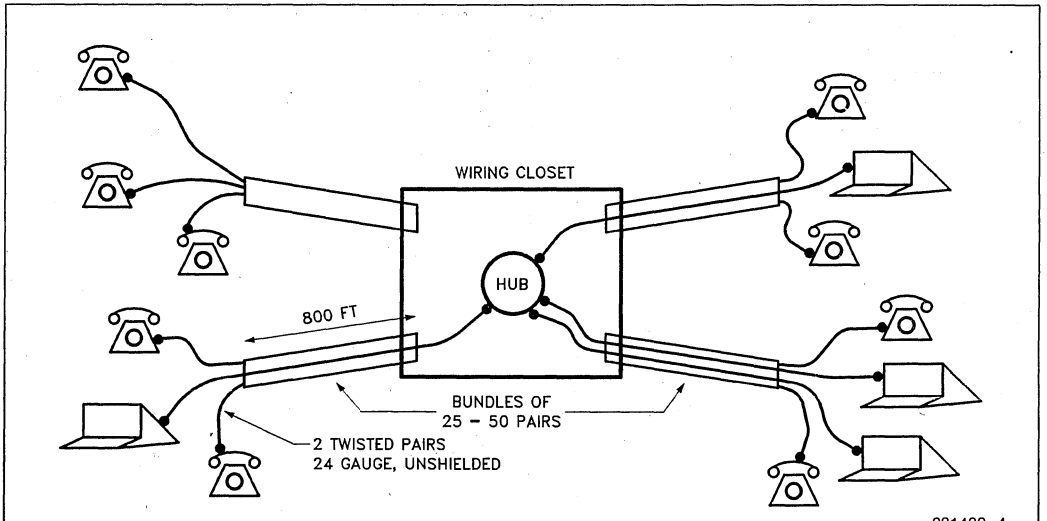


Figure 7-3. Telephone Wiring in a Building

building in the USA. Figure 7-3 shows how a typical building is wired for telephones. 24 gauge unshielded twisted pair wires emanate from a room called the Wiring Closet. The wires are in bundles of 25 or 50 pairs. The bundle is called D inside wiring (DIW) cable. The wires in these cables end up at modular telephone jacks in the wall. The telephone set is either connected directly to the jack or through an extension cable. Each telephone generally needs one twisted pair for voice and one more for auxiliary power. Thus, each modular jack has 2 twisted pairs (4 wires) connected to it. A 25 pair DIW cable can thus be used for up to 12 telephone connections. In most buildings, all pairs in a cable are not used up. Typically, a cable is used for only 4 to 8 telephone connections. This practice is followed by telephone companies because it is cheaper to install extra wires once, than to install once again to expand the existing number of connections. As a result, a lot of extra, unused wiring exists in a building. The stretch of cable between the wiring closet and the telephone jack is typically less than 800 ft. (250 meters). In the wiring closet the incoming wires from the telephones are routed to another wiring closet, a PABX or to the central office through an interconnect matrix. Thus, the wiring closet is a concentration point in the telephone network. There is also a redundancy of wires between the wiring closets.

7.2.1.2 StarLAN AND THE TELEPHONE NETWORK

Does StarLAN need telephone wiring in the building? Not really. StarLAN does not have to run on the building telephone wiring but the fact that it can, adds to its attractiveness. Figure 7-4 shows how the StarLAN network fits right on top of the telephone network. Each node needs 2 twisted pair wires to hook up to the HUB. The unused wires in the 25 pair DIW cables provide an electrical path up to the wiring closet, where the HUB is located. Note that the telephone and the StarLAN networks are electrically isolated. They only use the wires in the same DIW cable to reach the wiring closet. Within the wiring closet, the StarLAN wires go to a HUB and the telephone wires are routed to a different channel. Similar cable sharing can occur in going from one HUB to another. See Figure 7-5 for a typical office wired for StarLAN through the telephone wiring.



*StarLAN and telephones share the same cable, but are electrically isolated.
*StarLAN uses the unused wires in existing cables.

Figure 7-4. Coexistence of Telephone and StarLAN

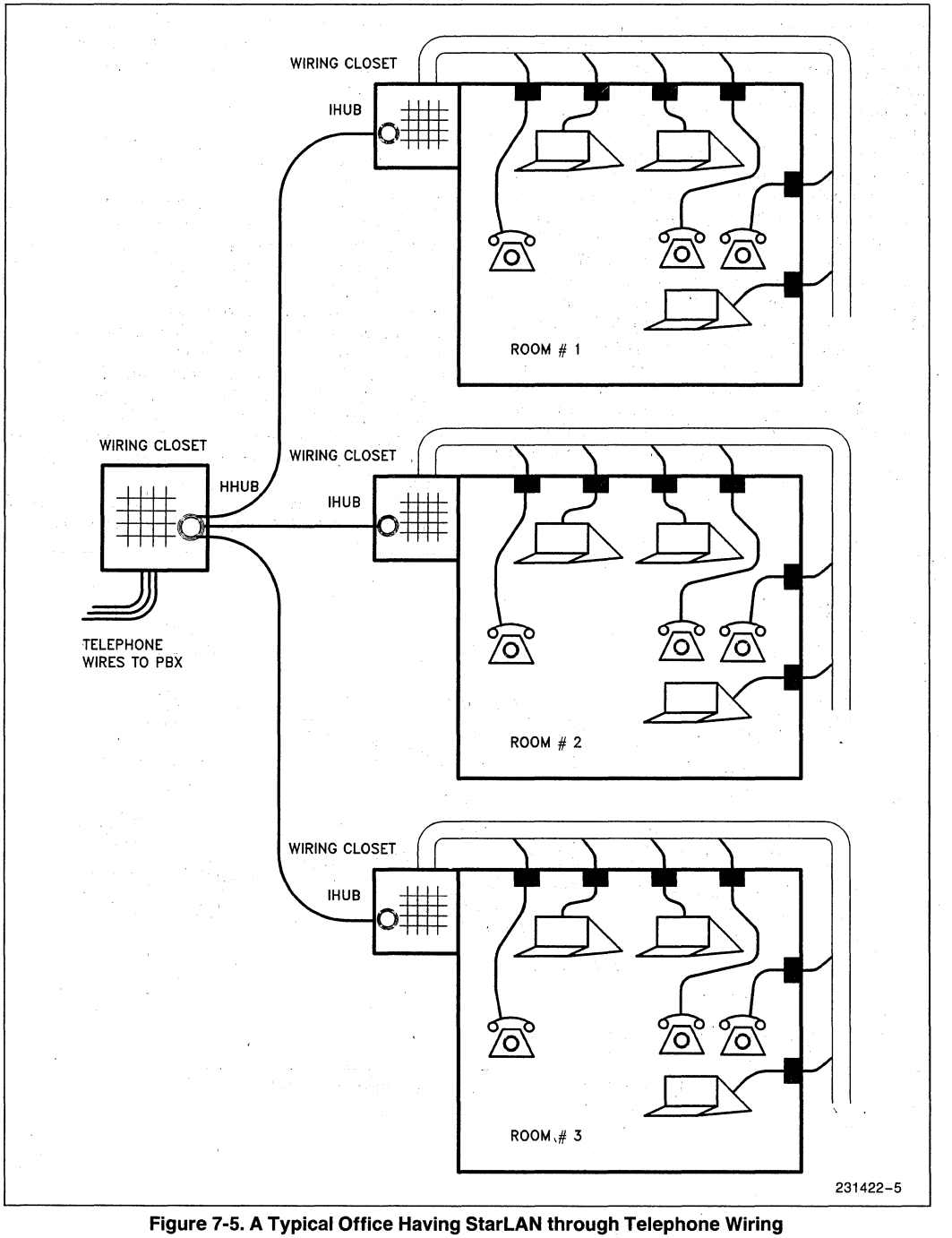


Figure 7-5. A Typical Office Having StarLAN through Telephone Wiring

7.2.2 StarLAN and Ethernet

Both StarLAN and Ethernet are CSMA/CD networks which conform to the IEEE 802.3 requirements. Since Ethernet has been around longer and is better understood, a comparison of Ethernet with StarLAN can ease the understanding of StarLAN.

a. Both Ethernet and StarLAN are IEEE 802.3 compatible CSMA/CD networks.

b. Data rate of Ethernet is 10Mb/s and that of StarLAN is 1 Mb/s.

c. Ethernet has a bus topology where each node is connected to a coaxial cable bus via a 50 meter transceiver cable containing four shielded twisted pair wires. StarLAN has a star topology, where each node is connected to a central HUB by a point to point link through two pairs of unshielded twisted pair wires.

d. Collision detection in Ethernet is done by the transceiver in the coaxial cable. Electrically, it is done by sensing the energy level on the coax cable. Collision detection in StarLAN is done in the HUB by sensing activity on all the input lines to the HUB.

e. In Ethernet, the presence of collision is conveyed by the transceiver to the node by a special collision detect (CDT) signal. In StarLAN, it is conveyed by the HUB using a special collision presence signal on the receive data line to the node.

f. Ethernet cable segments are interconnected using repeaters in a non-hierarchical fashion so that the distance between any two nodes does not exceed 2.5 kilometers. In StarLAN the maximum distance between two nodes is also 2.5 kilometers. This is achieved by wiring a maximum of five levels of HUBs in a hierarchical fashion.

It is interesting to see that topologically, Ethernet looks similar to a StarLAN, if the length of cable in Ethernet were to shrink to zero and the length of the transceiver cables were to grow to 800 ft. (250 meters), as shown in Figure 7-6.

7.2.3 Basic StarLAN Components

A StarLAN network has three basic components:

a. StarLAN node interface

b. StarLAN HUB

c. Cable

7.2.3.1 A StarLAN NODE INTERFACE

Figure 7-7 shows a typical StarLAN node interface. It interfaces to a processor on the system side. The processor runs the networking software. The heart of the node interface is the LAN controller which does the job of receiving and transmitting the frames in adherence to the IEEE 802.3 standard protocol. It maintains all

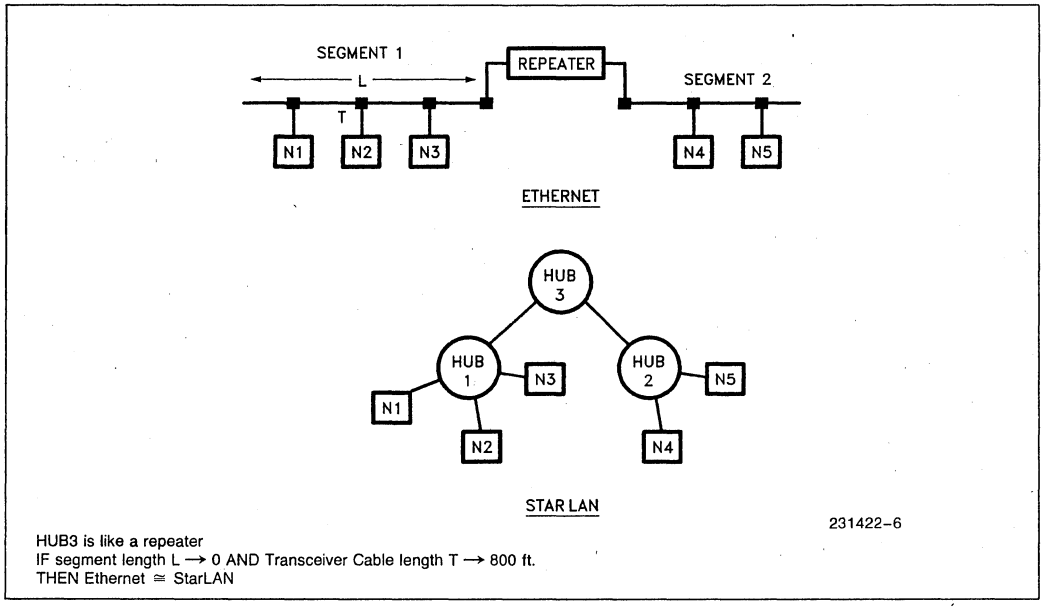


Figure 7-6. Ethernet and StarLAN Similarities

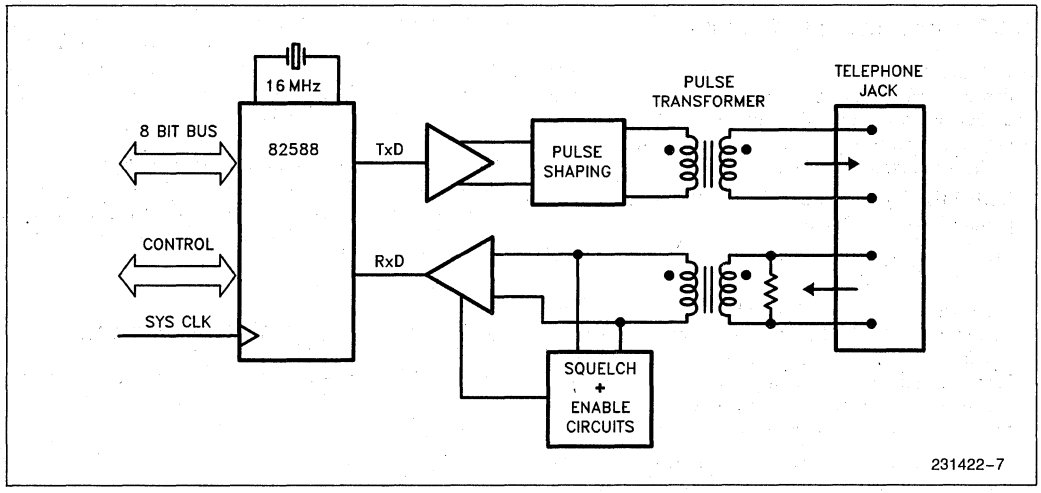


Figure 7-7. 82588 Based StarLAN Node

the timings—like the slot time, interframe spacing etc.—required by the network. It performs the functions of framing, deferring, backing-off, collision detection which are necessary in a CSMA/CD network. It also does Manchester encoding of data to be transmitted and clock separation— or decoding—of the Manchester encoded data that is received. The signals from the controller are converted to the differential form by a RS-422 or RS-485 driver. These signals cannot be directly sent on the unshielded twisted pair wire because the rise and fall times of the signal are fast and this causes the undesired effect of cross-talk and radiation. This disturbs other signals, digital and voice, sharing the same cable. Some pulse shaping is therefore done essentially to increase the rise and fall times (edges are made to rise and fall slower). The shaped signal is sent on to the twisted pair wire through a pulse transformer for DC isolation. The signals on the wire are thus differential, DC isolated from the node and almost sinusoidal (due to shaping and the capacitance of the wire).

The signal received by the node (from the HUB) is filtered from noise by a squelch circuit. The squelch circuit prevents idle line noise from affecting the receiver circuits in the LAN controller. The differential signal from the HUB is received using a zero-crossing RS-422 receiver. Output of the receiver, qualified by the squelch circuit, is fed to the RxD pin of the LAN controller. The RxD signal provides three kinds of information.

a. Normal received data, when receiving the frame.
b. Collision information in the form of the collision presence signal from the HUB. This is used when transmitting a frame.
c. Carrier sense information, indicating the beginning and the end of frame. This is useful during transmit and receive operations.

7.2.3.2 StarLAN HUB

HUB is the point of concentration in StarLAN. All the nodes transmit to the HUB and receive from the HUB. Figure 7-8 shows an abstract representation of the HUB. It has an upstream and a downstream signal processing unit. The upstream unit has N signal inputs and 1 signal output. And the downstream unit has 1 input and N output signals. The inputs to the upstream unit come from the nodes or from the intermediate HUBs (IHUBs) and its output goes to a higher level HUB. The downstream unit is connected the other way around; input from a upper level HUB and the outputs to nodes or lower level IHUBs. Physically each input and output consists of one twisted pair wire carrying a differential signal. The downstream unit essentially just re-times the signal received at the input, and sends it to all its outputs. The functions performed by the upstream unit are:

a. Collision detection
b. Collision Presence signal generation
c. Signal Retiming
d. Jabber Function

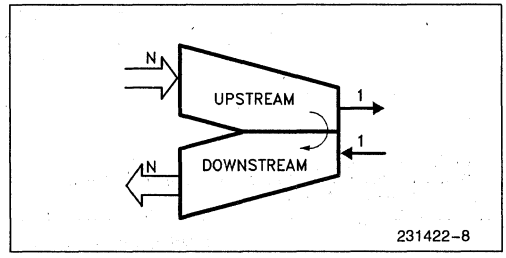


Figure 7-8. A StarLAN HUB

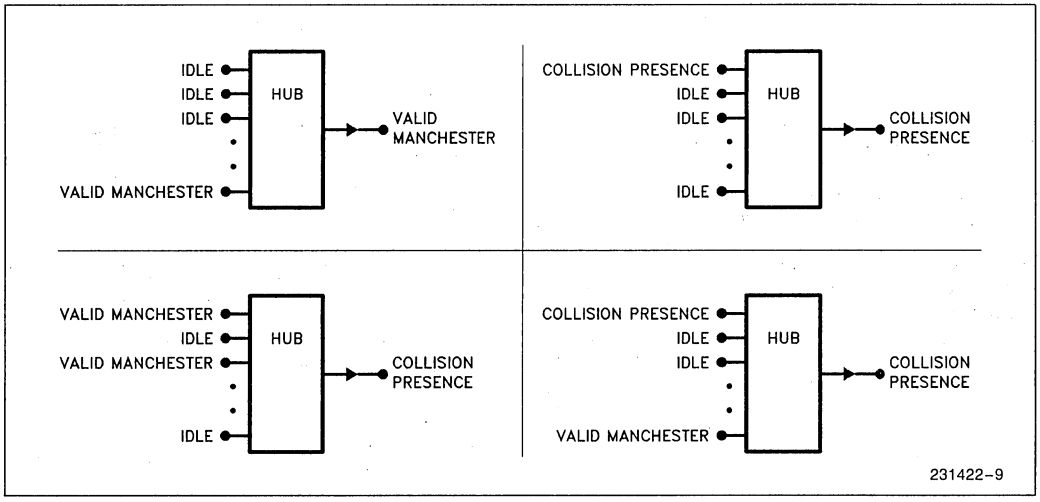


Figure 7-9. HUB as a Black Box

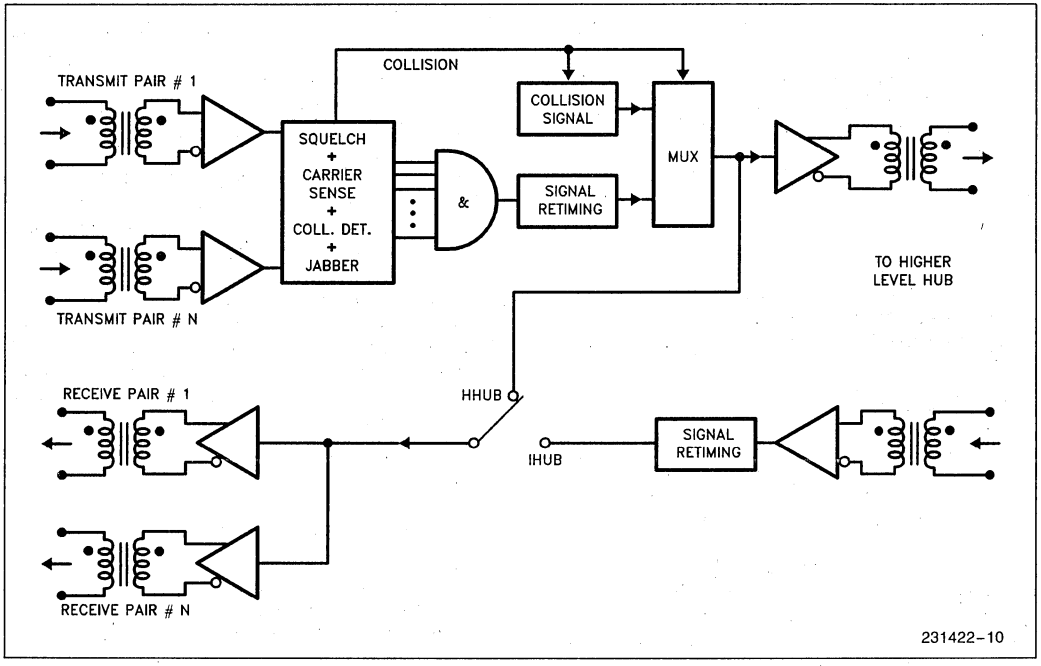


Figure 7-10. StarLAN HUB Block Diagram

The collision detection in the HUB is done by sensing the activity on the inputs. If there is activity (or transitions) on more than one input, it is assumed that more than one node is transmitting. This is a collision. If a collision is detected, a special signal called the Collision Presence Signal is generated. This signal is generated and sent out as long as activity is sensed on any of the input lines. This signal is interpreted by every node as an occurrence of collision. If there is activity only on one input, that signal is re-timed—or cleaned up of any accumulated jitter—and sent out. Figure 7-9 shows the input to output relations for the upstream part of the HUB as a black box.

If a node transmits for too long the HUB exercises a Jabber function to disable the node from interfering with traffic from other nodes. There are three timers in the HUB associated with this function and their operation is described in section 7.6.

Figure 7-10 shows a block diagram of the HUB. A switch position determines whether the HUB is an IHUB or a HHUB. If the HUB is an IHUB, the switch decouples the upstream and the downstream units. Header HUB (HHUB) is the highest level HUB; it has no place to send its output signal, so it returns its output signal (through the switch) to the outputs of the downstream unit. There is one and only one HHUB in a StarLAN network and it is always at the base of the tree. The returned signal eventually reaches every node in the network through the intermediate nodes (if any).

StarLAN specifications do not put any restrictions on the number of IHUBS at any level or on number of inputs to any HUB. The number of inputs per HUB are typically 10 to 20 and is dictated by the typical size of clusters in a given networking environment.

7.2.3.3 StarLAN CABLE

Unshielded telephone grade twisted pair wires are used to connect a node to a HUB or to connect two HUBs. This is one of the cheapest types of wire and responsible for bringing down the cost of StarLAN.

Although the 24 gauge wire is used for long stretches, the actual connection between the node and the telephone jack in the wall is done using extension cable, just like connecting a telephone to a jack. For very short StarLAN configurations, where all the nodes and the HUB are in the same room, the extension cable with plugs at both ends may itself be sufficient for all the wiring.

The telephone twisted pair wire of 24 gauge has the following characteristics:

Attenuation	: 42.55 db/mile @ 1 MHz
DC Resistance	: 823.69 Ω/mile
Inductance	: 0.84 mH/mile
Capacitance	: 0.1 μF/mile
Impedance	: 92.6Ω, −4 degrees @ 1 MHz

Experiments have shown that the sharing of the telephone cable with other voice and data services does not cause any mutual harm due to cross-talk and radiation, provided every service meets the FCC limits.

Although it is not a part of the IEEE 802.3 1BASE5 standard, there is considerable interest in using fiber optics and coaxial cable for node to HUB or HUB to HUB links especially in noisy and factory environments. Both these types of cables are particularly suited for point-to-point connections. Even mixing of different types of cables is possible.

7.2.4 Framing

Figure 7-11 shows the format of a StarLAN frame. The beginning of the frame is marked by the carrier going active and the end marked by carrier going inactive. The preamble has a 56 bit sequence of 101010 . . . . ending in a 0. This is followed by 8 bits of start of frame delimiter (sfd) − 10101011. These bits are transmitted with the MSB (leftmost bit) transmitted first. Source and destination fields are 6 bytes long. The first byte is the least significant byte. These fields are transmitted with LSB first. The length field is 2 bytes long and gives the length of data in the Information field. The entire information field is a minimum of 46 bytes and a maximum of 1500 bytes. If the data content of the Information field is less than 46, padding bytes are used to make the field 46 bytes long. The Length field indicates how much real data is in the Information field. The last 32 bits of the frame is the Frame Check Sequence (FCS) and contains the CRC for the frame. The CRC is calculated from the beginning of the destination address to the end of the Information field. The generating polynomial (Autodin II) used for CRC is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} +$$

$$X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

The frames can be directed to a specific node (LSB of address must be 0), to a group of nodes (multicast or group—LSB of address must be 1) or all nodes (broadcast—all address bits must be 1).

7.2.5 Signal Propagation and Collision

Figure 7-12 will be used to illustrate three typical situations in a StarLAN with two IHUBs and one HHUB. Nodes A and B are connected to HUB1, nodes C and D to HUB2 and node E to HUB3.

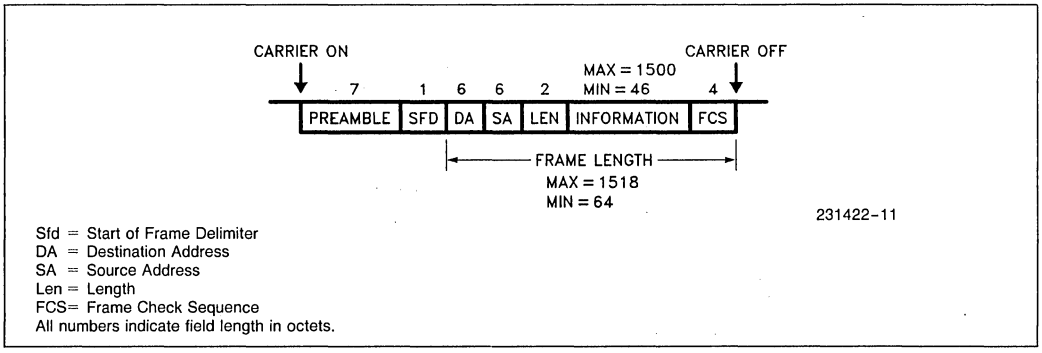


Figure 7-11. Framing

7.2.5.1 SITUATION #1

Whenever node A transmits a frame Fa, it will reach HUB1. If node B is silent, there is no collision. HUB1 will send Fa to HUB3 after re-timing the signal. If nodes C, D and E are also silent, there is no collision at HUB2 or HUB3. Since HUB3 is the HHUB, it sends the frame Fa to HUB1, HUB2 and to node E after re-timing. HUB1 and HUB2 send the frame Fa to nodes A, B and C, D. Thus, Fa reaches all the nodes on the network including the originator node A. If the signal received by node A is a valid Manchester signal and not the Collision Presence Signal (CPS) for the entire duration of the slot time, then the node A assumes that it was a successful transmission.

7.2.5.2 SITUATION #2

If both nodes A and B were to transmit, HUB1 will detect it as a collision and will send signal Fx (the Collision Presence Signal) to the HUB3—Note that HUB1 does not send Fx to nodes A and B yet. HUB 3 receives a signal from HUB1 but nothing from node E or HUB2, thus it does not detect the situation as a collision and simply re-times the signal Fx and sends it to node E, HUB2 and HUB1. Fx ultimately reach all the nodes. Nodes A and B detect this signal as CPS and call it a collision.

7.2.5.3 SITUATION #3

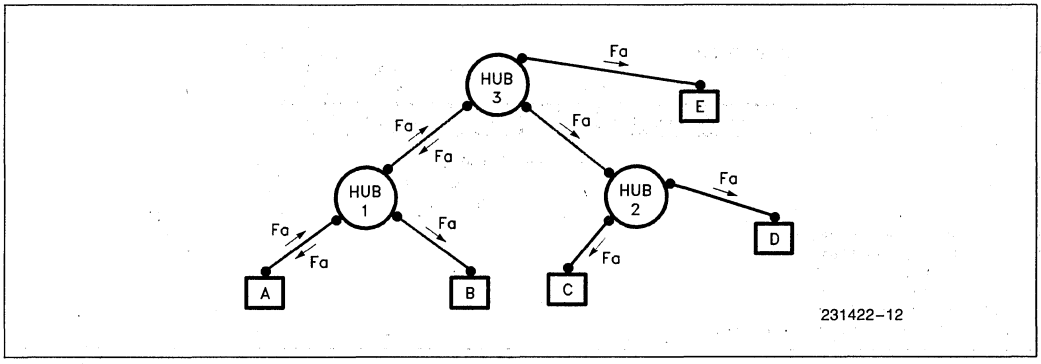
In addition to nodes A and B, if node C were also to transmit, the situation at HUB1 will be the same as in situation #2. HUB2 will propagate Fc from C towards HUB3. HUB3 now sees two of its inputs active and hence generates its own Fx signal and sends it towards each node.

These situations should also illustrate the point made earlier in the chapter that, the StarLAN network, with nodes connected to multiple HUBs is, in effect, equivalent to all the nodes connected to a single HUB.

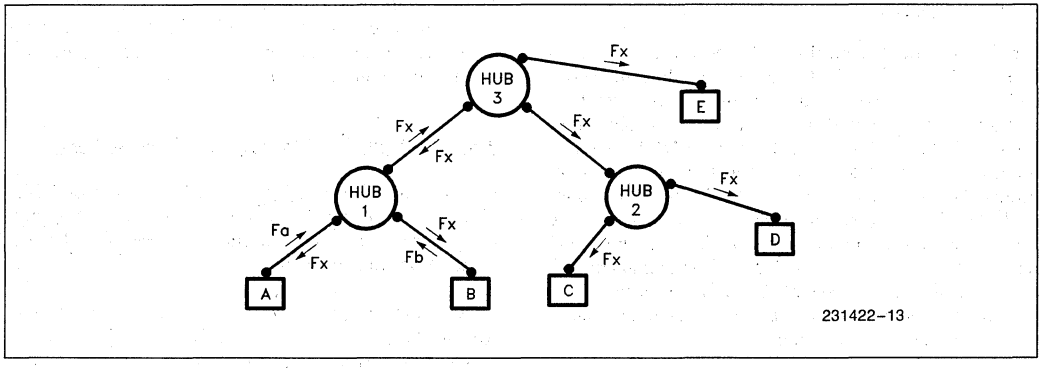
7.2.6 StarLAN Network Parameters

At the time of writing (June, 1985 revision of IEEE 802.3-1BASE5 specifications), all the StarLAN network parameters defined, match those of Ethernet. Some important ones are:

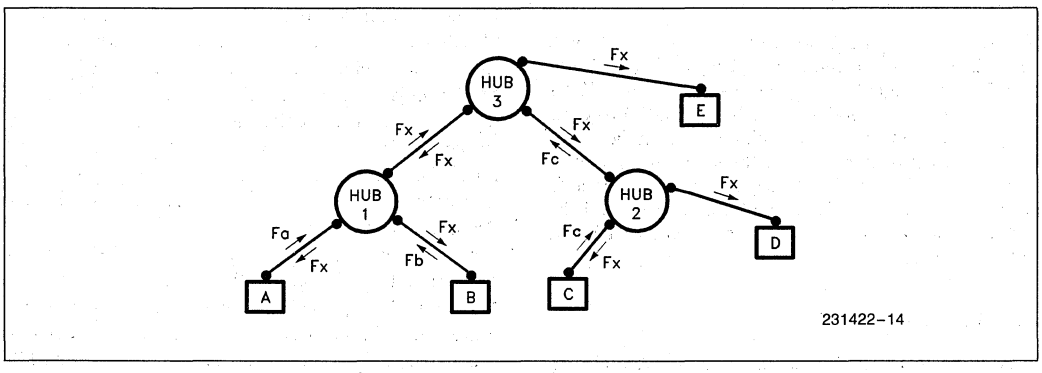
Preamble length (incl. sfd)	64 bits
Address length	6 bytes
FCS length CRC(Autodin II)	32 bits
Maximum frame length	1518 bytes
Minimum frame length	64 bytes
Slot time	512 bit times
Interframe spacing	96 bit times
Minimum jam timing	32 bit times
Maximum number of collisions	16
Backoff limit	10
Backoff method	Truncated binary exponential
Encoding	Manchester
Propagation delay between most distant nodes	130 bit times
Maximum delay though IHUB	10 bit times
Maximum delay per cable segment	4 bit times
Bits eaten up in the HUB	4 bits
Clock tolerance	±0.01%
Maximum jitter per segment	±90 ns



Situation #1. A Transmitting



Situation #2. A & B Transmitting



Situation #3. A, B & C Transmitting

HUB1, HUB2 are IHUBs
HUB3 is the HHUB

Fa, Fb, Fc—Frames from nodes A, B & C
Fx—Collision Presence Signal

Figure 7-12. Signal Propagation and Collisions

7.3 LAN CONTROLLER FOR StarLAN

One of the attractive features of StarLAN is the availability of the 82588, a VLSI LAN controller, designed to meet the needs of a StarLAN node. The main requirements of a StarLAN node controller are:

1. IEEE 802.3 compatible CSMA/CD controller.
2. Configurable to StarLAN network and system parameters.
3. Generation of all necessary clocks and timings.
4. Manchester data encoding and decoding.
5. Detection of the Collision Presence Signal.
6. Carrier Sensing.
7. Squelch or bad signal filtering.
8. Fast and easy interface to the processor.

82588 performs all these functions in silicon, providing a minimal hardware interface between the system processor and the StarLAN physical link. It also reduces the software needed to run the node, since a lot of functions, like deferring, back off, counting the number of collisions etc., are done in silicon.

7.3.1 IEEE 802.3 Compatibility

The CSMA/CD control unit on the 82588 performs the functions of deferring, maintaining the Interframe Space (IFS) timing, reacting to collision by generating a jam pattern, calculating the back-off time based on the number of collisions and a random number, decoding the address of the incoming frame, discarding a frame that is too short, etc. All these are performed by the 82588 in accordance to the IEEE 802.3 standards. For inter-operability of different nodes on the StarLAN network it is very important to have the controllers strictly adhere to the same standards.

7.3.2 Configurability of the 82588

Almost all the networking parameters are programmable over a wide range. This means that the StarLAN parameters form a subset of the total potential of the 82588. This is a major advantage for networks whose standards are being defined and are in a flux. It is also an advantage in carrying over the experience gained with the component in one network to other applications, with differing parameters.

The 82588 is initialized or configured to its working environment by the CONFIGURE command. After the execution of this command, the 82588 knows its system and network parameters. A configure block in memory is loaded into the 82588 by DMA. This block contains all the parameters to be programmed as shown in Figure 7-13. Following is a partial list of the parame-

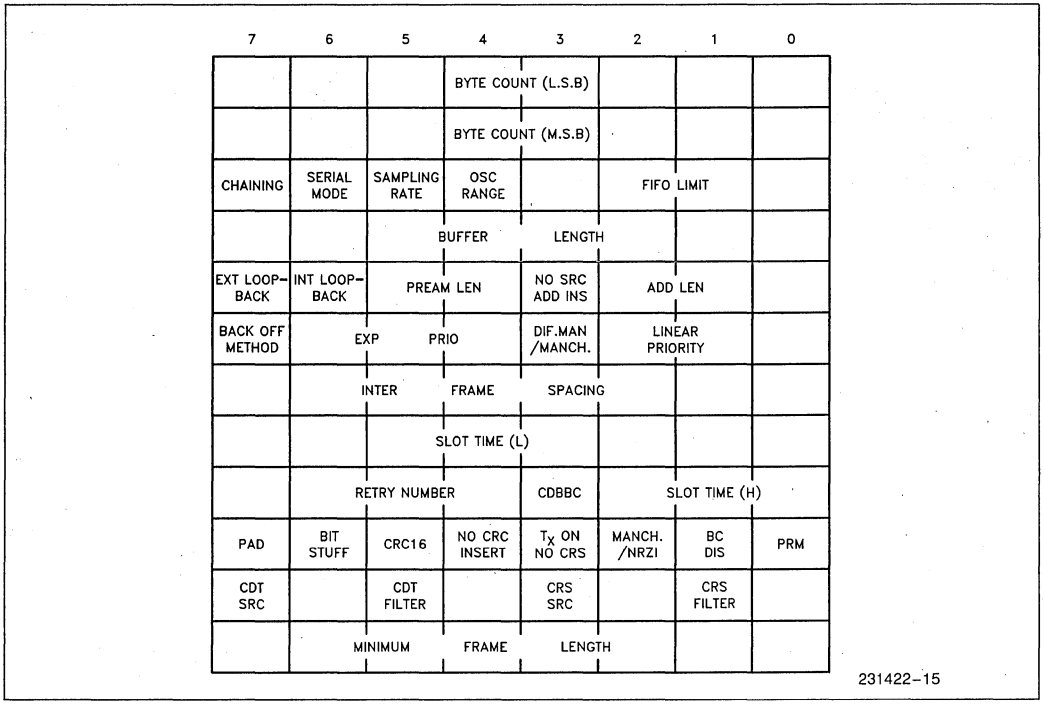


Figure 7-13. Configuration Block

ters with the programmable range and the StarLAN value:

Parameter	Range	StarLAN Value
Preamble length	2, 4, 8, 16 bytes	8
Address length	0 to 6 bytes	6
CRC type	16, 32 bit	32
Minimum frame length	6 to 255 bytes	64
Interframe Spacing	12 to 255 bit times	96
Slot time	1 to 2047 bit times	512
Number of retries	0 to 15	15
Data encoding	NRZI, Man., Diff. Man.	Manch.
Collision detection	Code viol., Bit comp.	Code Viol.

Beside these, there are many other options available, which may or may not apply to StarLAN:

Data sampling rate of 8 or 16
Operating in Promiscuous mode
Reception of Broadcast frames
Internal loopback operation
External loopback operation
Transmit without CRC
HDLC Framing
:
:

7.3.3 Clocks and Timers

The 82588 requires two clocks; one for the operation of the system interface and another for the serial side. Both clocks are totally asynchronous to each other. This permits transmitting and receiving frames at data rates that are virtually independent of the speed at which system interface operates.

The serial clock can be generated on chip using just an external crystal of a value 8 or 16 times the desired bit rate. An external clock may also be used.

The 82588 has a set of timers to maintain various timings necessary to run the CSMA/CD control unit. These are timings for the Slot time, Interframe spacing time, Back off time, Number of collisions, Minimum frame length, etc. These timers are started and stopped automatically by the 82588.

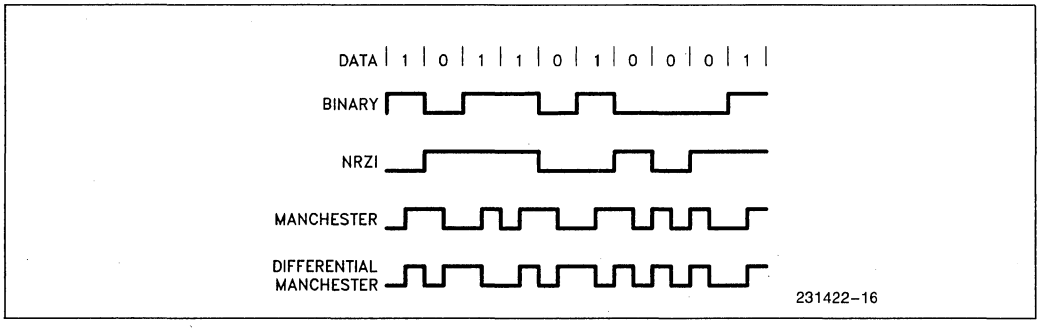
7.3.4 Manchester Data Encoding and Decoding

In StarLAN the data transmitted by the node must be encoded in Manchester format. Node should also be able to decode Manchester encoded data when receiving a frame—a process also known as clock recovery. The 82588 does the encoding and decoding of data bits for data rates up to 2 Mb/s.

Besides Manchester, the 82588 can also do encoding and decoding in NRZI and Differential Manchester formats. Figure 7-14 shows samples of encoding in these three formats. The main advantage of NRZI over the other two is that NRZI requires half the channel bandwidth, for any given data rate. On the other hand, since the NRZI signal does not have as many transitions as the other two, clock recovery from it is more difficult. The main advantage of Differential Manchester over straight Manchester is that for a signal that is differentially driven (as in RS 422), crossing of the two wires carrying the data does not change the data received at the receiver. In other words, NRZI and Differential Manchester encoding methods are polarity insensitive.

7.3.5 Detection of the Collision Presence Signal

In a StarLAN network, HUB informs the nodes that a collision has occurred by sending the Collision Presence Signal (CPS) to the nodes. The CPS signal is a special signal which contains violations in Manchester encoding. Figure 7-15 shows the CPS signal. It has a 5 microsec. period, looking very much like a valid Manchester signal except for missing transitions (or violations) at periodic intervals. When the 82588 decodes this signal, it fails to see mid-cell transitions repeatedly at intervals of 2.5 bit times and hence calls it a code violation. The edges of CPS are marked for illustration as a, b, c, d, . . . l. Let us see how the 82588 interprets the signal if it starts calling the edge 'a' as the mid-cell transition for '1'. Then edge at 'b' is '0'. Now the 82588 expects to see an edge at '*' but since there is none, it is a Manchester code violation. The edge that eventually does occur at 'd' is then used to re-synchronize and, since it is a falling edge, it is taken as a mid-cell transition for '0'. The edge at 'e' is for a '1' and then again there is no edge at '*'. This goes on, with the 82588 flagging code violation and re-synchronizing again every 2.5 bit times as shown in Figure 7-15. When a transmitting node sees this CPS signal being returned by the HUB (instead of a valid Manchester signal it transmitted), it assumes that a collision occurred. The 82588 has two built-in mechanisms to detect collisions. These mechanisms are very general and can be used for a very broad class of applications to detect collisions in



Encoding Method	Mid Bit Cell Transitions	Bit Cell Boundary Transitions
Binary	Do not exist.	Identical to original data.
NRZI	Do not exist.	Exist only if original data bit equals 0. Dependent on present encoded signal level: to 0 if 1 to 1 if 0
Manchester	Exist for every bit of the original data: from 0 to 1 for 1 from 1 to 0 for 0	Exist for consequent equal bits of original data: from 1 to 0 for 1 1 from 0 to 1 for 0 0
Differential Manchester	Exist for every bit of the original data. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0	Exist only if original data bit equals 0. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0

Figure 7-14. 82588 Data Encoding Rules

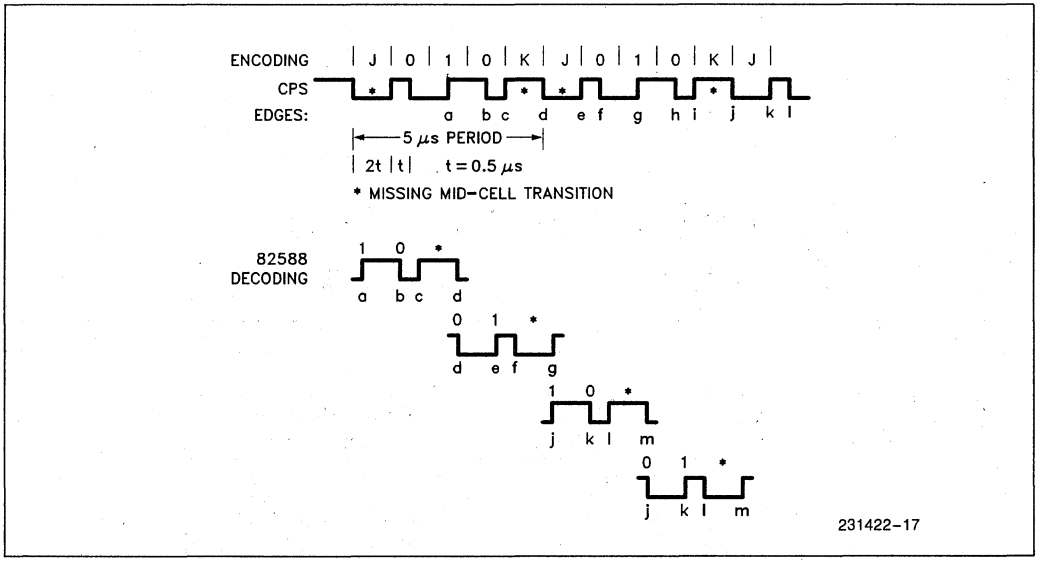


Figure 7-15. 82588 Decoding the Collision Presence Signal

a CSMA/CD network. Using these mechanisms, the 82588 can detect collisions (two or more nodes transmitting simultaneously) by just receiving the collided signal during transmission, even if there were no HUB generating the CPS signal.

7.3.5.1 COLLISION DETECTION BY CODE VIOLATION

If during transmission, the 82588 sees a violation in the encoding (Manchester, NRZI or Differential Manchester) used, then it calls it a collision by aborting the transmission and transmitting a 32 bit jam pattern. The algorithm used to detect collision, and even to do the data decoding, is based on finding the number of sampling clocks between an edge to the next edge. Suppose an edge occurred at time 0, the sampling instant of the next edge determines whether it was a collision (C), a long pulse (L)—with a nominal width of 1 bit time—or a short pulse (S)—nominal width of half a bit time. The following two charts show the decoding and collision detection algorithm for sampling rates of 8 and 16 when using Manchester encoding. The numbers at the bottom of the line indicate sampling instances after the occurrence of the last edge (at 0). The alphabets on the top show what would be inferred by the 82588 if the next edge were to be there.

Sampling rate = 8 (clock is 8x bit rate)

```
C  C  S  S  S  L  L  L  L  L  C  C
|--|--|--|--|--|--|--|--|--|--|--|--|--|
0  1  2  3  4  5  6  7  8  9  10 11 12 13
```

Collision also if:

RxD stays low for 13 samples or more

A mid cell transition is missing

Sampling rate = 16 (clock is 16x bit rate)

```
C C C C C S S S S S S C L L L L L L L L L C C C C
|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
0  2  4  6  8  10 12 14 16 18 20 22 24 26
```

Collision also if:

RxD stays low for 13 samples or more

A mid cell transition is missing

A single instance of code violation can qualify as collision. The 82588 has a parameter called collision detect filter (CDT Filter) that can be configured from 0 to 7. This parameter determines for how many bit times the violation must remain active to be flagged as a collision. For StarLAN CDT Filter must be configured to 0—that is disabled.

7.3.5.2 COLLISION DETECTION BY SIGNATURE (OR BIT) COMPARISON

This method of collision detection compares a signature of the transmitted data with that of the data received on the RxD pin while transmitting. Figure 7-16 shows a block diagram of the logic. As the frame is transmitted it flows through the CRC generation logic. A timer, called the Tx slot timer, is started at the same time that

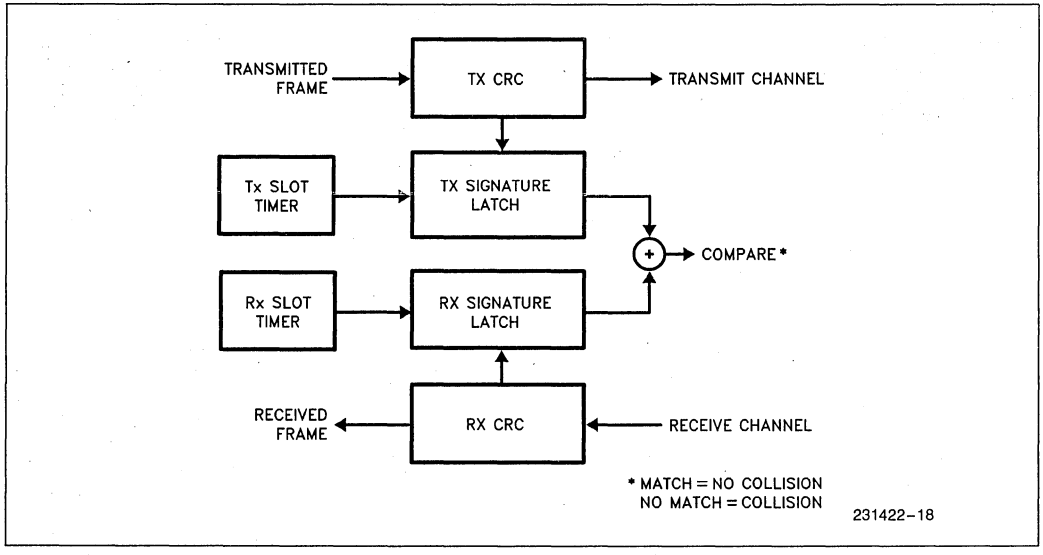


Figure 7-16. Collision Detection by Signature Comparison

the CRC generation starts. When the count in the timer reaches the slot time value, the current value of the CRC generator is latched in as the transmit signature. As the frame is returned back (through the HUB) it flows through the CRC checker. Another timer—Rx slot timer—is started at the same time as the CRC checker starts checking. When this timer reaches the slot time value, the current value of the CRC checker is latched in as the receive signature. If what is received is same as what was transmitted during the collision window, then it is assumed that there was no collision. Whereas, if the signatures do not match, a collision is assumed to have occurred.

Note that, even if the collision were to occur in the first few bits of the frame, a slot time must elapse before it is detected. In the code violation method, collision is detected within a few bit times. However, since the signature method compares the signatures, which are characteristic of the frame being transmitted, it is more robust. The code violation method can be fooled by returning a signal to the 82588 which is not the same as the transmitted signal but is a valid Manchester signal—like a 1 MHz signal. Both methods can be used simultaneously giving a combination of speed and robustness.

7.3.5.3 ADDITIONAL COLLISION DETECTION MECHANISM

In addition to the collision detection mechanisms described in the preceding sections, the 82588 also flags collision when after starting transmission any of these conditions become valid:

a. Half a slot time elapse and the carrier sense of 82588 is not active.

b. Half a slot time + 16 bit times elapse and the opening flag (sfd) is not detected.

c. Carrier sense goes inactive after an opening flag is received with transmitter still active.

These add a further robustness to the collision detection mechanism of the 82588. It is also possible to OR an externally generated collision detect signal to the internally generated condition.

7.3.6 Carrier Sensing

StarLAN network is considered to be busy if there are transitions on the cable. Carrier is supposed to be active if there are transitions. Every node controller needs to know when the carrier is active and when not. This is done by the carrier sensing circuitry. On the 82588 this circuit is on chip. It looks at the RxD (receive data) pin and if there are transitions, it turns on an internal carrier sense signal. It turns off the carrier sense signal if RxD remains in idle (high) state for 1.6 bit times. This carrier sense information is used to mark the start of the interframe space time and the back off time. The 82588 also defers transmission when the carrier sense is active.

When operating in the NRZI encoded mode, carrier sense is turned off if RxD pin is in the idle state for 8 bit times (instead of 1.6) or more.

7.3.7 Squelching the Input

Squelch circuits are used to filter out bad signal on the receive data input. Two types of filtering are necessary. One in the voltage domain—called the voltage squelch, another in the time domain—called the time squelch. Squelch improves the reliability of the node and also the stability of the network.

Voltage quelch is done to filter out signal whose strength is below a defined threshold (0.6 volts for StarLAN). It prevents idle line noise from disturbing the receive circuits on the controller. The voltage squelch circuit is placed right after the receiving pulse transformer. It enables the input to the RxD pin to the 82588 only when the signal strength is above the threshold.

If the signal received has the proper level but not the proper timing, it should not bother the receiver. This is accomplished by the time squelch circuit on the 82588. Time squelching is essential to weed out spikes, glitches and bad signal especially at the beginning of a frame. The 82588 does not turn on its carrier sense (or receive enable) signal until it receives three consecutive edges, each separated by time periods greater than $\frac{1}{8}$th bit times at x16 sampling (and $\frac{1}{4}$ bit times at x8 sampling) but less than 1.6 bit times as shown in Figure 7-17. See how spikes are filtered out.

The carrier sense activation can be programmed for a further delay by up to 7 bit times by a configuration parameter called carrier sense filter. See Figure 7-17.

7.3.8 System Bus Interface

The 82588 has a conventional bus interface making it very easy to interface it to any processor bus. Figure 7-18 shows that it has an 8 bit data bus, read, write, chip select, interrupt and reset pins going to the processor bus. It also needs an external DMA controller for data transfer. A system clock of up to 8 MHz is also needed. The read and write access times of the 82588 are very short—95 ns—as shown by Figure 7-19. This further facilitates interfacing the controller to almost any processor.

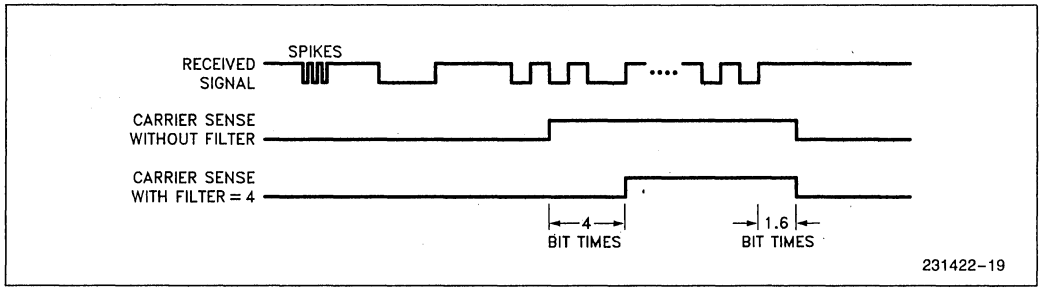


Figure 7-17. Carrier Sensing and Squelch

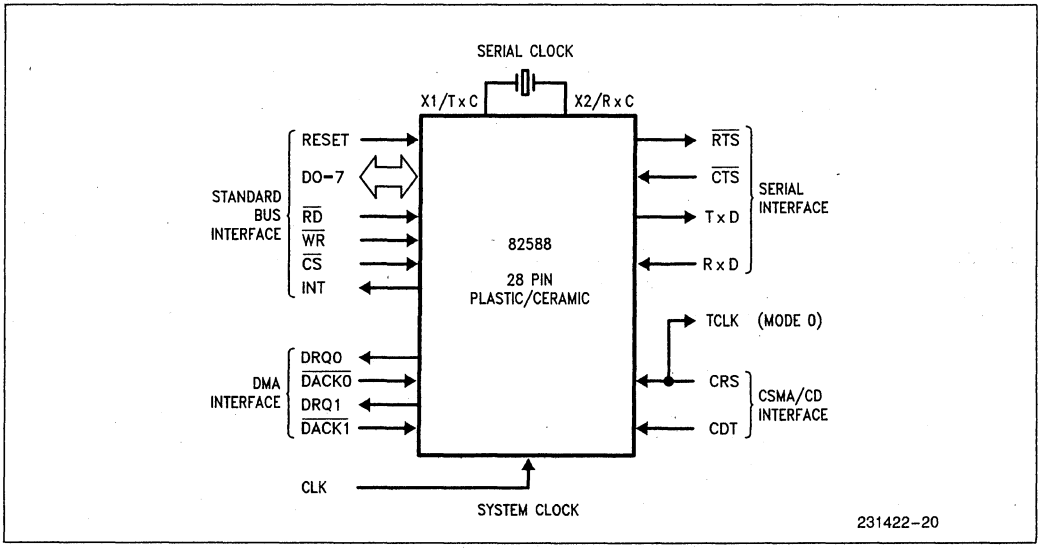


Figure 7-18. Chip Interface

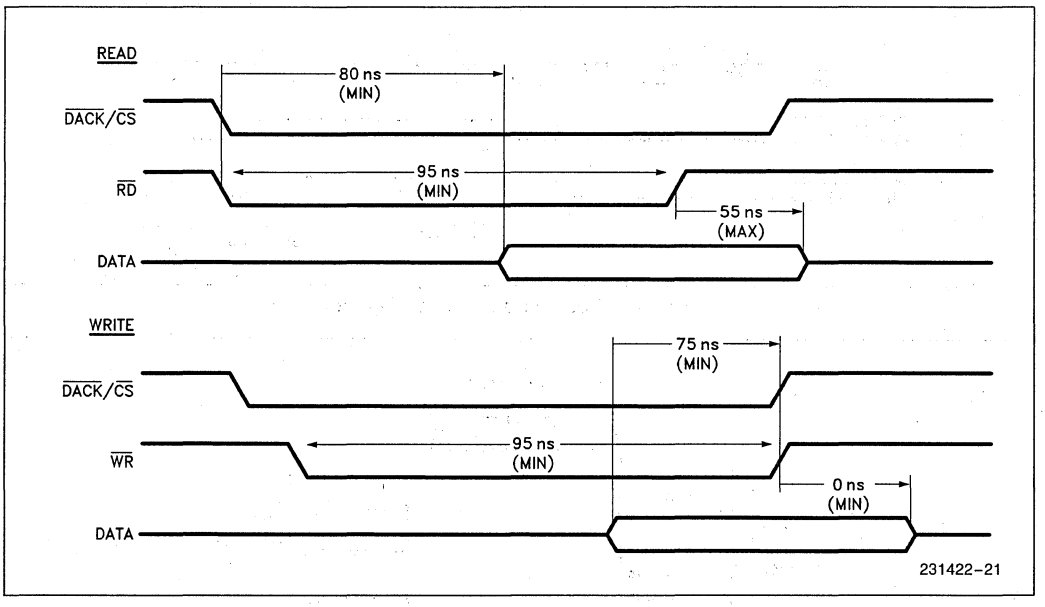


Figure 7-19. Access Times

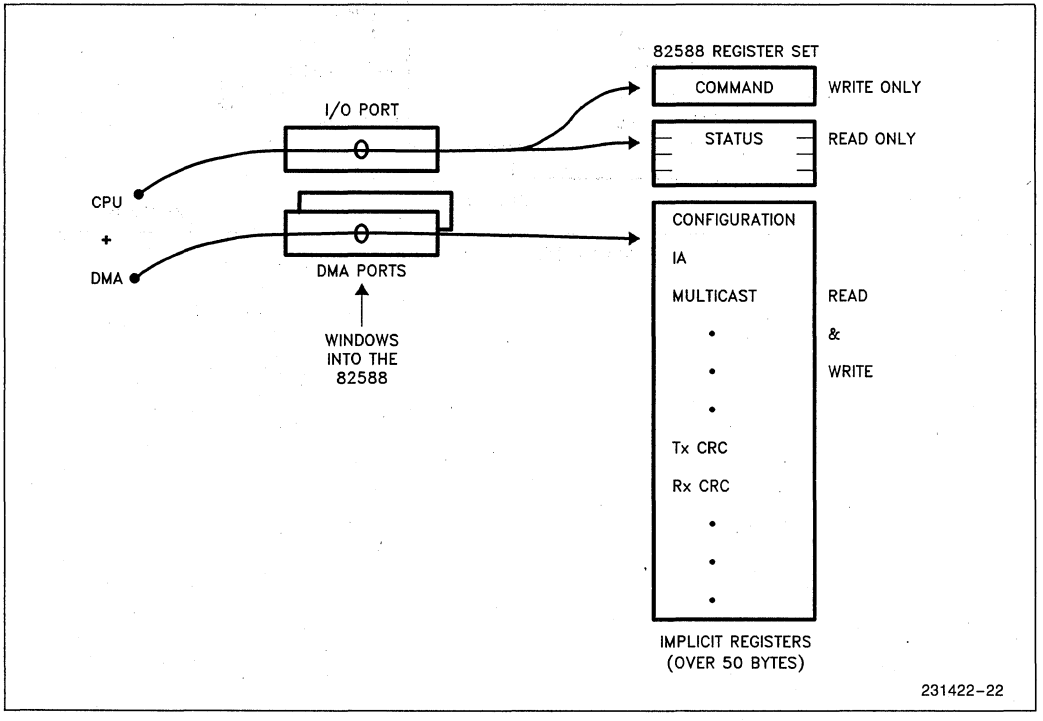
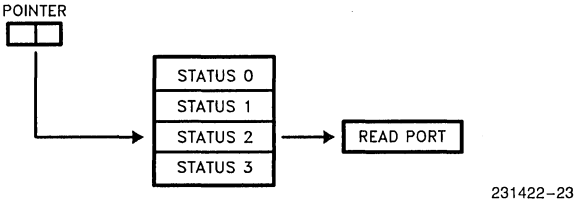


Figure 7-20. Register Access

4 Status registers are accessed through one read port



231422-23

The pointer can be changed using a command or can be automatically incremented.

```
READ_STATUS_588: PROCEDURE;                    /* COMMAND 15 */

     OUTPUT (CS_588) = 15;              /* RELEASE POINTER, INITIAL = 00 */

     STATUS_588(0)=INPUT (CS_588);      /* REFRESH STATUS REGISTER IMAGE */

     STATUS_588(1)=INPUT (CS_588);      /* IN MEMORY.

     STATUS_588(2)=INPUT (CS_588);

     STATUS_588(3)=INPUT (CS_588);

     RETURN

END READ_STATUS_588;
```

READING 4 STATUS REGISTERS

Figure 7-21. Reading the Status Register

The 82588 has over 50 bytes of registers, and most are accessed only indirectly. Figure 7-20 shows the register access mechanism of the 82588. It has one I/O port and 2 DMA channel ports. These are the windows into the 82588 for the CPU and the DMA controller. An external CPU can write into the Command register and read from the Status registers using I/O instructions and asserting chip select and write or read lines. Although there is just one I/O port and 4 status registers, they can be read out in a round robin fashion through the same port as shown in Figure 7-21. Other registers like the Configuration, Individual Address registers can be accessed only through DMA. All the internal registers can be dumped into memory by DMA using the Dump command. The execution of some of the commands is described in section 7.4. See the 82588 Reference Manual for details on these commands.

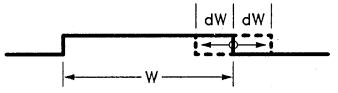
7.3.9 Debug and Diagnostic Aids

Besides the standard functions that can be used directly for StarLAN, the 82588 offers many debug and diagnostics functions. The DIAGNOSE command of the 82588 does a self-test of most of the counters and timers in the 82588 serial unit. Using the DUMP command, all the internal registers of the 82588 can be dumped into the memory. The TDR command does Time Domain Reflectometry on the network. The 82588 has two loopback modes of operation. In the internal loopback mode the 82588 can receive its own transmitted frame. This is very useful to test the transmit and receive units of the chip and also the system interface. The external loopback can be used to test even the external link at the full data rate.

7.3.10 Jitter Performance

When the 82588 receives a frame from the HUB, the signal has a jitter. The jitter is the shifting of the edges of the signal from the nominal position due to the transmission over a length of cable. Many factors like, intersymbol (resulting due to specific sequence of 0's and 1's) interference, rise and fall times of drivers and receivers, cross talk, etc., contribute to the jitter. StarLAN specifies a maximum jitter of ± 90 ns whenever the signal goes from a node/HUB to HUB/node. Figure 7-22 shows that the jitter tolerance of the 82588 is 120 ns for Manchester encoded data at 1 Mb/s giving an ample safety margin.

$$\text{Jitter} = \frac{\pm \text{ variation in pulse width}}{\text{Nominal Pulse width}} = \frac{dW}{W}$$



231422-24

Manchester Encoded Data:

±25% ..... (23%)
±125 ns for a 500 ns pulse ..... (120 ns)
±250 ns for a 1000 ns pulse ..... (240 ns)

NRZI Encoded Data:

±4.2% ..... (4%)
±250 ns for a 6000 ns pulse ..... (240 ns)
±25% ..... (23%)
±250 ns for a 1000 ns pulse ..... (240 ns)

**Numbers in parenthesis are practical values.

Figure 7-22. 82588 Jitter Tolerance

7.4 THE 82588

This chapter describes the basic 82588 operations. Please refer to the 82588 Reference Manual or the LAN Components User's Manual for a detailed description. Basic operations like transmitting a frame, receiving a frame, configuring the 82588 and dumping the register contents are discussed here to give a feel for how the 82588 works.

7.4.1 Transmit and Retransmit Operations

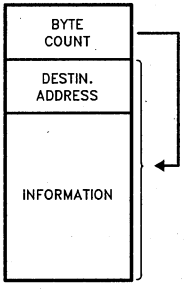
To transmit a frame, the CPU prepares a block in the memory called the transmit data block. As shown in Figure 7-23, this block starts with a byte count field, indicating how long the rest of the block is. The destination address field contains the node address of the destination. Rest of the block contains the information or the data field of the frame. The CPU also programs the DMA controller with the start address of the transmit data block. The DMA byte count must be equal to or greater than the block length. The 82588 is then issued a TRANSMIT command—an OUT instruction to the command port of the 82588. The 82588 starts generating DMA requests to read in the transmit data block by DMA. It also determines whether and how long it must defer on the link and when it can, it starts transmitting with the preamble. The 82588 constructs the frame on the fly. It takes the destination address from the memory, source address as its own individual address (previously programmed), data field from the memory and the CRC, generated on chip, at the end of the frame.

At the conclusion of transmission the 82588 generates an interrupt to the CPU. The CPU can read the status registers to find out if the transmission was successful. If a collision occurs during transmission, the 82588 aborts transmission and generates the jam sequence, as required by IEEE 802.3, and informs the CPU by interrupt and the status register. It also starts the back-off timer.

To re-attempt transmission, the CPU must reinitialize the DMA controller to the start of the transmit data block and issue a RETRANSMIT command to the 82588. When the 82588 receives the retransmit command and the back-off timer has expired, it transmits again. Interrupt and the status register contents again indicate the success or failure of the (re)transmit attempt.

The main difference between transmit and retransmit command is that retransmit command does not clear the internal count for the number of collision occurred, whereas transmit command does. Moreoever, retransmit takes effect only when the back-off timer has expired.

1. Prepare Transmit Data—Block in Memory
2. Program DMA Controller
3. Issue Transmit Command on the Desired Channel



231422-25

Transmit Data Block

4. Interrupt is received on completion of command or if the command was aborted or there was a collision. The status bytes 1 and 2 indicate the result of the operation.

7	6	5	4	3	2	1	0	
TX DEF	HRT BEAT	MAX COLL		NUM. OF COLLISIONS				STATUS 1
COLL		TX OK			LOST CRS	LOST CTS	UNDER RUN	STATUS 2

231422-26

Transmit & Retransmit Results Format

Figure 7-23. Transmit Operation

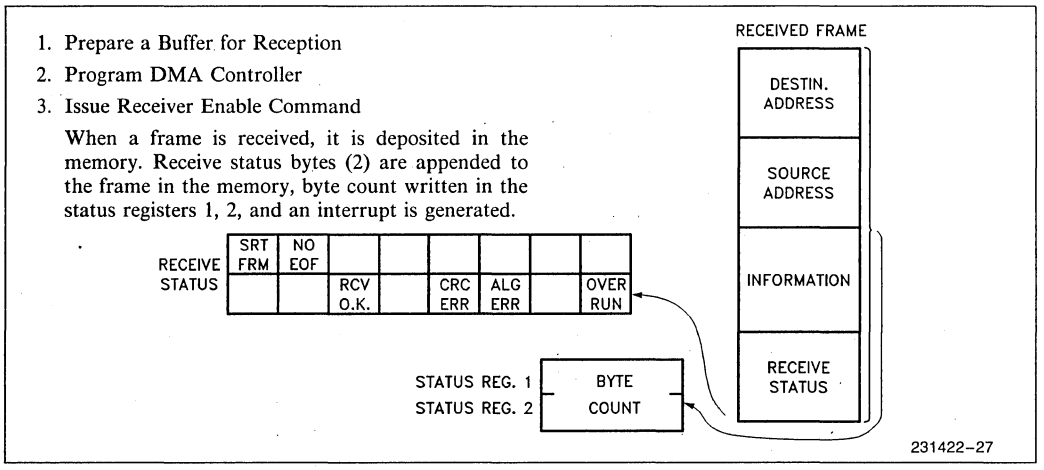


Figure 7-24. Receive Operation (Single Buffer)

7.4.2 Configuring the 82588

To initialize the 82588 and program its network and system parameters, a configure operation is performed. It is very similar to the transmit operation. Instead of a transmit data block as in transmit command, a configure data block—shown in Figure 7-13—is prepared by the CPU in the memory. The first two bytes of the block specify the length of rest of the block, which specify the network and system parameters for the 82588. The DMA controller is then programmed by the CPU to the beginning of this block and a CONFIGURE command is issued to the 82588. The 82588 reads in the parameters by DMA and loads the parameters in the on-chip registers.

Similarly, for programming the INDIVIDUAL ADDRESS and MULTICAST ADDRESSes, the DMA controller is used to load the 82588 registers.

7.4.3 Frame Reception

Before enabling the 82588 for reception the CPU must make a buffer available for the frame to be received. The CPU must program the DMA controller with the starting address of the buffer and then issue the ENABLE RECEIVER command to the 82588. When a frame arrives at the RxD pin of the 82588, it starts receiving the frame. Only if the address in the destination address matches either the Individual address, Multicast address or if it is a broadcast address, is the frame deposited into memory by the 82588 using DMA. The format of storage in the memory is shown in Figure 7-24. At the end, a two byte field is attached which shows the status of the received frame. If CRC, alignment or overrun errors are encountered, they are reported. An interrupt from 82588 occurs when all the bytes have been transferred to the memory. This informs the CPU that a new frame has been received.

If the received frame has errors, the CPU must recover (or re-use) the buffer. Note that the entire frame is deposited into one buffer.

7.4.3.1 MULTIPLE BUFFER FRAME RECEPTION

It is also possible to receive a frame into a number of fixed size buffers. This is particularly economical if the received frames vary widely in size. If the single buffer scheme were used as described above, the buffer required would have to be bigger than the longest expected frame and would be very wasteful for very short (typically acknowledge or control) frames. The multiple buffer reception is illustrated in Figure 7-25. It uses two DMA channels for reception.

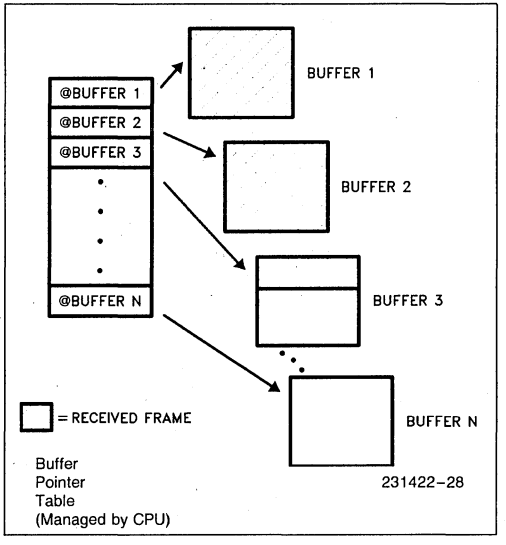


Figure 7-25. Multiple Buffer Reception

As in single buffer reception, the one channel, say channel 0, of the DMA controller is programmed to the start of buffer 1, and the 82588 is enabled for reception with the chaining bit set. As soon as the first byte is read out of the 82588 by the DMA controller and written into the first location of buffer 1, the 82588 generates an interrupt, saying that it is filling up its last available buffer and one more buffer must be allocated. The filling up of the buffer 1 continues. The CPU responds to the interrupt by programming the other DMA channel—channel 1—with the start address of the second buffer and issuing an ASSIGN ALTERNATE buffer command with an INTACK (interrupt acknowledge). This informs the 82588 that one more buffer is available on the other channel. When buffer 1 is filled up (the 82588 knows the size of buffers from the configuration command), the 82588 starts generating the DMA requests on the other channel. This automatically starts filling up buffer 2. As soon as the first byte is written into buffer 2, the 82588 interrupts the CPU again asking for one more buffer. the CPU programs the channel 0 of the DMA controller with the start address of buffer 3, issues an ASSIGN ALTERNATE buffer command with INTACK. This keeps the buffer 3 ready for the 82588. This switching of channels continues until the entire frame is received generating an end of frame interrupt. The CPU maintains the list of pointers to the buffers used.

Since a new buffer is allocated at the time of filling up of the last buffer. The 82588 automatically switches to the new buffer to receive the next frame as soon as the last frame is completely received. It can start receiving the new frame almost immediately even before the end of frame interrupt is serviced and acknowledged by the CPU. If a new frame comes in, and the previous frame interrupt is not yet acknowledged, the interrupt line goes active again for the buffererd one.

If by the time a buffer fills up no new buffer is available, the 82588 keeps on receiving. An overrun will occur and will be reported in the received frame status. However, ample time is available for the allocation of a new buffer. It is roughly equal to the time to fill up a buffer. For 128 byte buffers it is 128 × 8 = 1024 microseconds or approximately 1 millisec. You get 1 ms to assign a new buffer after getting the interrupt for it. Hence the process of multiple buffer reception is not time critical for the system performance.

This method of reception is particularly useful to guarantee the reception of back-to-back frames separated by IFS time. This is because a new buffer is always available for the new frame after the current frame is received.

Although both the DMA channels get used up in receiving, only one channel is kept ready for reception and the other one can be used for other commands until the reception starts. If an execution command like transmit or dump command is being executed on a channel which must be allocated for reception, the command gets aborted when the ASSIGN ALTERNATE BUFFER command is issued to the channel used for the execution command. The interrupt for command aborted occurs after the end of frame interrupt.

7.4.4 Memory Dump of Registers

All the 82588 internal registers can be dumped in the memory by the DUMP command. A DMA channel is used to transfer the register contents to the memory. It is very similar to reception of a frame; instead of data from the serial link, the data from the registers gets written into the memory. This provides a software debugging and diagnostic tool.

7.4.5 Other Operations

Other 82588 operations like DIAGNOSE, TDR, ABORT, etc. do not require any parameter or data transfer. They are executed by writing a command to the 82588 command register and knowing the results (if any) through the status registers.

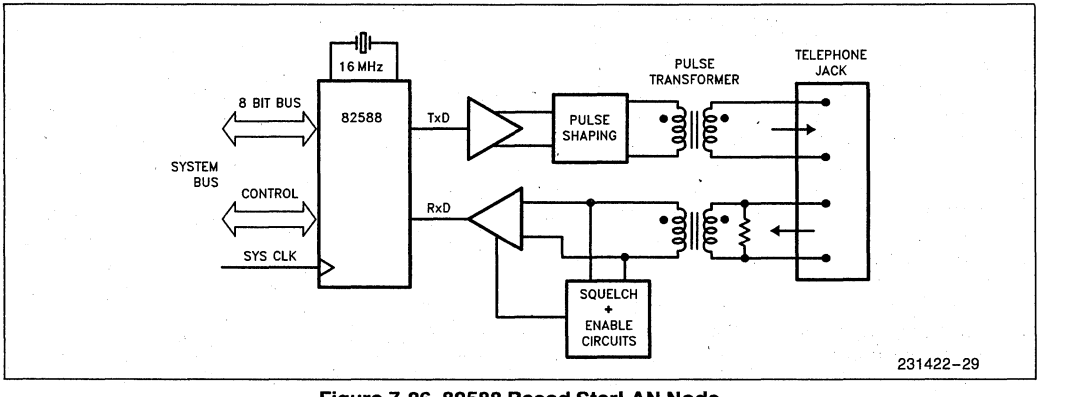


Figure 7-26. 82588 Based StarLAN Node

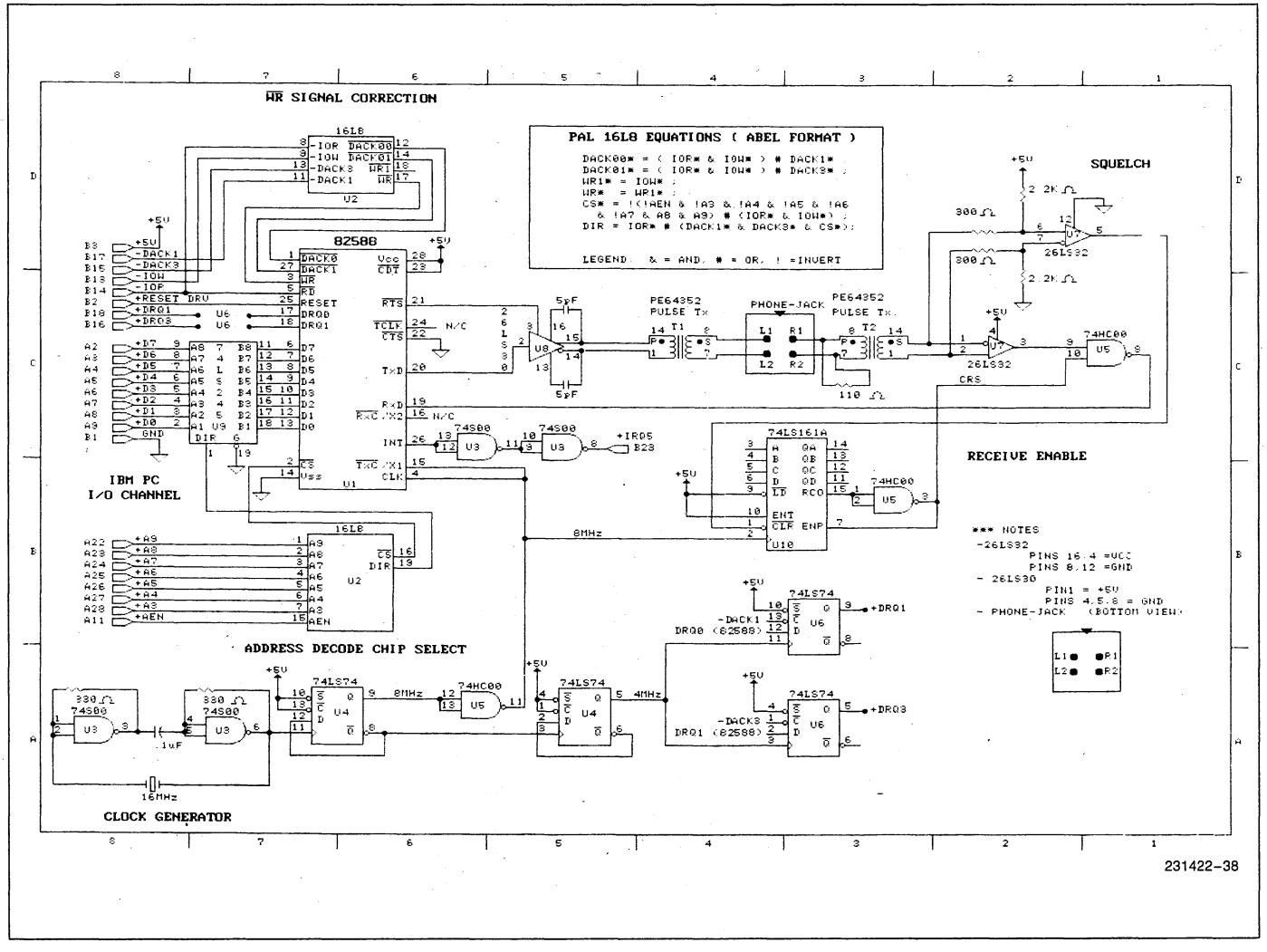


Figure 7-27. StarLAN Node Schematics

7.5 StarLAN NODE FOR IBM PC

This chapter deals with the hardware—the StarLAN board—to interface the IBM PC to a StarLAN Network. This is a slave board which takes up one slot on the I/O channel of the IBM PC. Figure 7-26 shows an abstract block diagram of the board. It requires the IBM PC resources of the CPU, memory, DMA and interrupt controller on the system board to run it. Such a board has two interfaces. The IBM PC I/O Channel on the system or the parallel side and the telephone grade twisted pair wire on the serial side. Figure 7-27 shows the circuit diagram of the board.

7.5.1 Interfacing to the IBM PC I/O Channel

IBM PC has 8 slots on the system board to allow expansion of the basic system. All of them are electrically identical and the I/O channel is the bus that links them all to the 8088 system bus. The I/O channel contains an 8 bit bidirectional data bus, 20 address lines, 6 levels of interrupt, 3 channels of DMA control lines and other control lines to do I/O and memory read/write operations. Figure 7-28 shows the signals and the pin assignment for the I/O Channel.

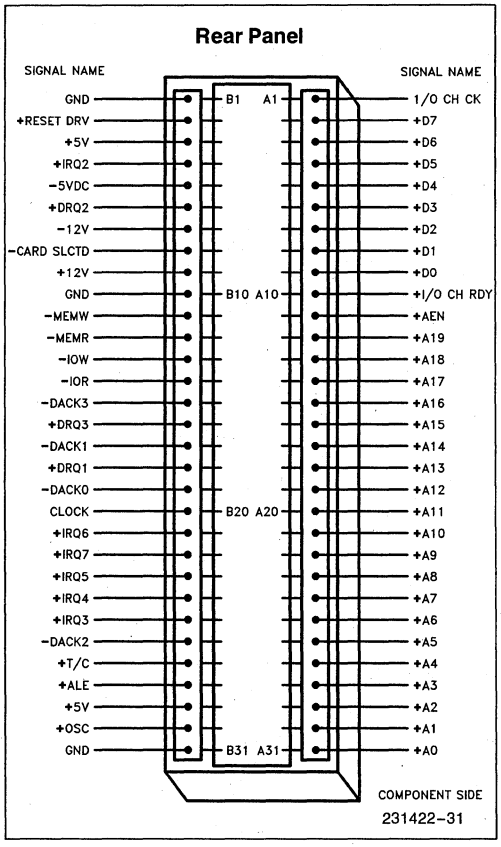


Figure 7-28. I/O Channel Diagram

7.5.1.1 CHIP SELECT AND DATA BUS INTERFACING

The 82588 on our board has to be accessible to the CPU on the system board. The CPU access the 82588 by I/O instructions. On the StarLAN board, chip select must be generated to select the 82588 when it is addressed. Figure 7-29 shows the I/O address map for the

Hex Range	Usage
000-00F	DMA Chip 8237A-5
020-021	Interrupt 8259A
040-043	Timer 8253-5
060-063	PPI 8255A-5
080-083	DMA Page Registers
0AX*	NMI Mask Register
0CX	Reserved
0EX	Reserved
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asynchronous Communications (Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk
378-37F	Printer
380-38C**	SDLC Communications
380-389**	Binary Synchronous Communications (Secondary)
3A0-3A9	Binary Synchronous Communications (Primary)
3B0-3BF	IBM Monochrome Display/Printer
3C0-3CF	Reserved
3D0-3DF	Color/Graphics
3E0-3E7	Reserved
3F0-3F7	Diskette
3F8-3FF	Asynchronous Communications (Primary)

* At power-on time, the Non Mask Interrupt into the 8088 is masked off.
This mask bit can be set and reset through system software as follows:
Set mask: Write hex 80 to I/O Address hex A0 (enable NMI)
Clear mask: Write hex 00 to I/O Address hex A0 (disable NMI)

** SDLC Communications and Secondary Binary Synchronous Communications cannot be used together because their hex addresses overlap.

Figure 7-29. I/O Address Map

IBM PC. Address of 300H was chosen for the StarLAN board. A PAL (16L8) is used to do the control signal interfacing between the 82588 and the I/O Channel. Signals A3 to A9 and AEN are used to generate the chip select for the 82588:

```
CS* = !(!AEN & !A3 & !A4 & !A5 & !A6 & !A7 & A8 & A9)
      # (IOR* & IOW*);
```

NOTE:
ABEL PAL programming language is used for PAL equations in this and the next section. An asterisk (*) following a signal name indicates that it is active low. Following operators are used:

! = invert (complement), # = logical OR, & = logical AND

The data bus D0 to D7 is buffered from the 82588 by 74LS245. The transceiver is always kept enabled. The direction of the transceiver is switched whenever a read operation is done by the CPU OR THE DMA controller using the equation:

```
DIR = IOR* # (DACK1* & DACK3* & CS*);
```

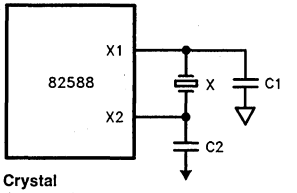
A part of the PAL (first 4 equations) is used to correct a problem with the timing of WR and DACK signals which is relevant only to the A-2 stepping of the 82588. B-step will not require the correction, although it will also work with this circuit.

7.5.1.2 CLOCK GENERATION

The 82588 requires two clocks for operation. The system clock and the serial clock. The serial clock can be generated on chip by putting a crystal across X1 and X2 pins. Alternatively, an externally generated clock can be fed in at pin X1 (with X2 left open). In both cases, the frequency must be either 8 or 16 times (sampling factor) the desired bit rate. For StarLAN, 8 or 16 MHz are the correct values to generate 1 Mb/s data rate. A configuration parameter is used to tell the 82588 what the sampling factor is. An externally supplied clock must have MOS levels (0.6V–3.9V). Specifications for the crystal and the circuit are shown in Figure 7-30.

X (MHz)	C1 (pf)	C2 (pf)
1–8	0–30	0–40
8–16	0	0–10

At 16 MHz, operation with C1 = C2 = 0 pf show no problems



231422-32

Recommended Crystal
Fundamental mode operation
Max Effective Series Resistance (ESR) = 30Ω
±0.005% tolerance @ 25 °C
±0.01% tolerance for 0–70 °C
Manufacturer CRYSTEK claims to satisfy these specifications.

Figure 7-30. Crystal Specs

The system clock has to be supplied externally. It can be up to 8 MHz. This clock runs the parallel side of the 82588. Its frequency does not have any impact on the read and write access times but on the rate at which data can be transferred to and from the serial side of the 82588. For the A-2 stepping, this clock must be a MOS level signal. For the B-stepping, a TTL level signal (0.8V–2.0V) will suffice.

The I/O channel of the IBM PC supplies a 4.77 MHz signal of 33% duty cycle. This would do for the system clock. It was decided to generate a separate clock on the StarLAN board to be independent of the I/O channel clock so that this board can also be used in future IBM PCs and also in some other compatibles. The 8 MHz clock is converted to MOS level by 74HC00 and fed into both the system and serial clock inputs.

7.5.1.3 DMA INTERFACE

The 82588 requires two DMA channels for full operation. In this application, one channel is dedicated for reception and the other is used to do transmit and the other commands. Could you get away if you had just one DMA channel available? Although using the IEEE 802.3 protocol you either transmit or receive but not both simultaneously, if a channel is not dedicated to reception, you may lose a frame if you had just one DMA channel and were about to use it for transmitting. Such a lost frame can only be recovered at a higher level of communications software. So the recommendation is not to operate with just one DMA channel. It is, however, possible to operate without losing frames and using just one DMA channel. Appendix B describes this method.

The IBM PC system board has one 8237A DMA controller. Channel 0 is used for doing the refresh of DRAMs. Channels 1, 2 and 3 are available for add-on boards on the I/O Channel. The floppy disk controller board uses the DMA channel 2 leaving exactly two channels (1 and 3) for the 82588. The situation is worse if the IBM PC/XT is used, since it uses channel 3 for the Winchester hard disk leaving just the channel 1 for the 82588. On the other hand, the IBM PC/AT has 5 free DMA channels even after the floppy and the hard disk consume one each. We will assume that 8237A DMA channels 1 and 3 are available for the 82588 as in the case of the IBM PC.

Since the channel 0 of 8237A is used to do refresh of DRAMs all the channels should be operated in single byte transfer mode. In this mode, after every transfer for any channel the bus is granted to the current high-

est priority channel. In this way, no channel can hog the bus bandwidth and, more important, the refresh of DRAMs is assured every 15 microseconds since the refresh channel (number 0) has the highest priority. This mode of operation is very slow since the HOLD is dropped by the 8237A and then asserted again after every transfer. Demand mode of operation is a lot more suitable to 82588 but it cannot be used because of the refresh requirements. Flip-flops are used to interface the DRQ lines from the 82588 to the I/O channel to cut off the DRQ after every transfer. This prevents the 8237A from locking up if the 82588 releases the DRQ line after the transfer has occurred having held it active for the duration of the transfer. It also prevents the interference to the refresh timing if the 8237A were programmed in the demand mode for the 82588.

7.5.1.4 INTERRUPT CONTROLLER

The 82588 interrupts the CPU after the execution of a command or on reception of a frame. It uses the 8259A interrupt controller on the system board to interrupt the CPU. There are 6 interrupt request lines, IRQ2 to IRQ7, on the I/O channel. Figure 7-31 shows the assignment of the lines. In fact, none of the lines are free for use. To add any new peripheral which uses a system board interrupt you have to see that the board that normally uses that interrupt is not being used. It was decided to use IRQ5 for the 82588. The INT signal from the 82588 is buffered and connected to IRQ5.

Number	Usage
NMI	Parity
0	Timer
1	Keyboard
2	Reserved
3	Asynchronous Communications (Secondary) SDLC Communications BSC (Secondary)
4	Asynchronous Communications (Primary) SDLC Communications BSC (Primary)
5	Fixed Disk
6	Diskette
7	Printer

Figure 7-31. IBM PC Hardware Interrupt Listing

7.5.2 Serial Link Interface

The StarLAN board is connected to the twisted pair wiring using an extension cable (up to 8 meters—25 ft.). See Figure 7-32. One end of the cable plugs into the telephone modular jack on the StarLAN board and the other end into a modular jack in the wall. The twisted pair wiring starts at the modular jack in the wall and goes to the wiring closet. In the wiring closet, another telephone extension cable is used to connect to a StarLAN HUB. The transmitted signal from the 82588 reach the on-board telephone jack through a RS-422 driver with pulse shaping and a pulse transformer. The received signals from the telephone jack to the 82588 come through pulse transformer, squelch circuit and a receive enable circuit.

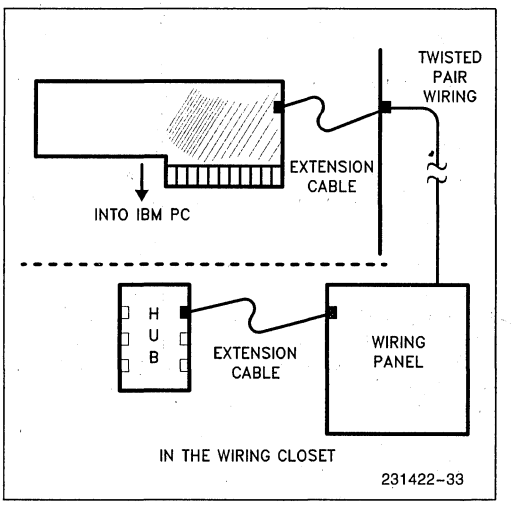
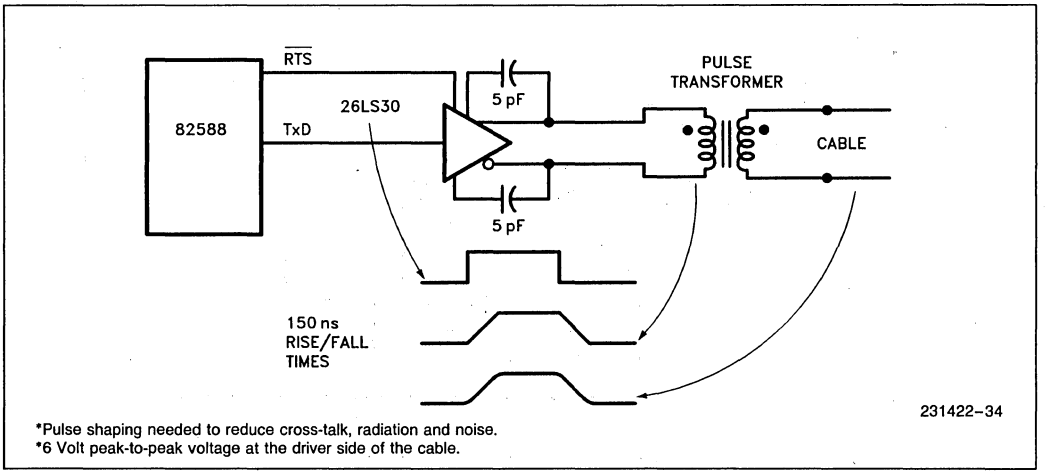


Figure 7-32. Path from StarLAN Board to HUB

7.5.2.1 TRANSMIT PATH

The single ended transmit signal on the TxD pin has to be converted to a differential signal, for noise immunity, and the rise and fall times increased to 150 to 200 nanoseconds before feeding it to the pulse transformer. Am26LS30 is a RS-422 driver which converts the TxD signal to a differential signal. It also has slew rate control pins to increase to rise and fall times. A large rise and fall time is a key requirement for operation at 1 Mb/s on telephone grade wires to cut out cross-talk, interference and radiation. The 26LS30 converts a square pulse to a trapezoidal one—see Figure 7-33. The filtering effect of the cable further adds to reduce the higher frequency components from the waveform so that on the cable the signal is almost sinusoidal. The pulse transformer is for DC isolation. Pulse transformers from Pulse Engineering—type PE 64352—are specially designed for StarLAN. They introduce an additional rise and fall time of about 70–100 ns on the signal. Dual pulse transformers in 14 pin DIP are manufactured under the part number PE 64382.



*Pulse shaping needed to reduce cross-talk, radiation and noise.
*6 Volt peak-to-peak voltage at the driver side of the cable.

Figure 7-33. Wave Shaping

7.5.2.2 RECEIVE PATH

The signal coming from the HUB over the twisted pair wire is received on the StarLAN board through a 110Ω line termination resistor and a pulse transformer. The pulse transformer is of the same type as for the transmit side and its function is dc isolation. The received signal which is differential and almost sinusoidal is fed to the Am26LS32 RS-422 receiver. As seen from the Figure 7-27 the pulse transformer feeds two RS-422 receivers. The one on the top is for squelch filtering and the one below is the real receiver which does real zero crossing detection on the signal and regenerates a square digital waveform from the sinusoidal signal that is received. Proper zero crossing detection is very essential; if the edges of the regenerated signal are not at zero crossings, the resulting signal may not be a proper Manchester encoded signal even if the original signal is valid Manchester. The resistors in the upper receiver keep its differential inputs at a voltage difference of 600 mV. These bias resistors ensure that the output of the upper receiver remains high as long as the input signal is less than 600 mV. It is very important that the RxD pin remains HIGH (not LOW or floating) whenever the receive line is idle. A violation of this may cause the 82588 to lock-up on transmitting. Remember, that based on the signal on the RxD pin, the 82588 extracts information on the data being received, Carrier Sense and Collision Detect. This squelch of 600 mV keeps the idle line noise from getting to the 82588. Figure 7-34 shows that when the differential input of the receiver crosses zero, a transition occurs at the output. It also shows that if the signal strength is below 600 mV, the output does not change. Note that the differential voltage at the lower receiver input is zero when the line is idle. The output of the squelch goes to a pulse stretcher which, as shown in Figure 7-35, generates an envelope of the received frame. The envelope is a receive enable and is used to AND the signal from the real zero crossing receiver before feeding it to the RxD pin of the 82588. RxD pin requires a MOS level input for the A-2 stepping of the 82588 hence 74HC00 is used to interface the receive signal to the 82588. For the B-stepping RxD will be a TTL level input.

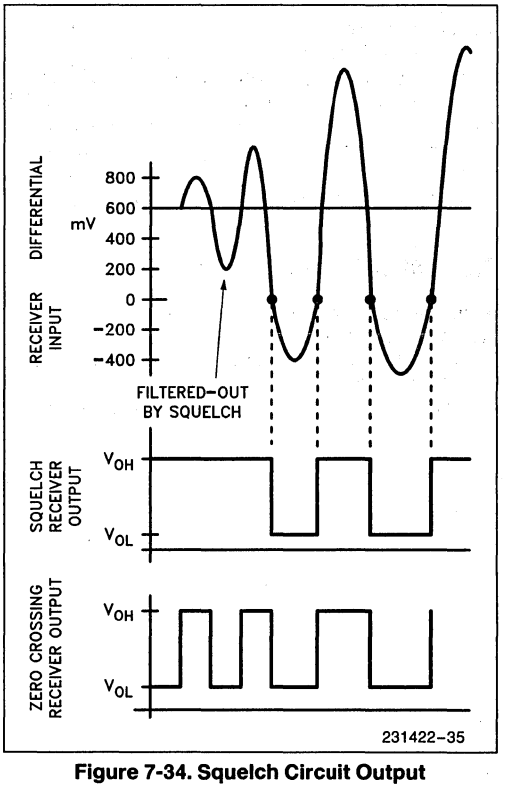
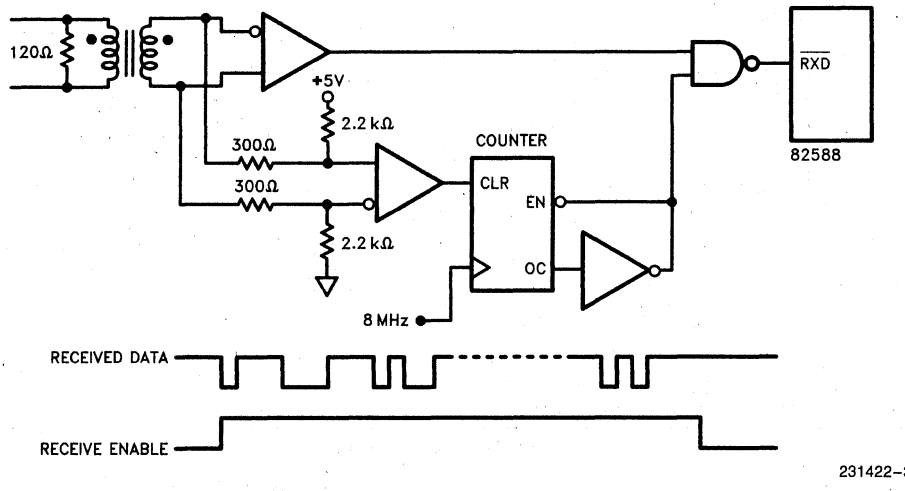


Figure 7-34. Squelch Circuit Output



*Squelch circuit suppresses noise on an idle line.
*0.6 volt threshold recommended for squelch.

Figure 7-35. Receiver Circuit

7.5.3 Cost

The parts needed for the circuit used cost about $70 in lk quantity in 1985. It occupies a board area of about 9 sq. inches (60 sq. cm). Beside the 82588, 2 pulse transformers, one receiver, one driver, one PAL and 5 SSI TTL chips are needed. A telephone modular jack and some passive components are also needed. Note that 3 of the 5 SSI chips would not be needed if the StarLAN interface were to sit on the motherboard.

7.5.4 80188 Interface to 82588

Although the 82588 interfaces easily to almost any processor, no processor offers as much of the needed functionality as the 80186 or its 8 bit cousin, the 80188. The 80188 is 8088 object code compatible processor with DMA, timers, interrupt controller, chip select logic, wait state generator, ready logic and clock generator functions on chip. Figure 7-36 shows how the 82588, in a StarLAN environment interfaces to the 80188. It uses the clock, chip select logic, DMA channels, interrupt controller directly from the 80188. The interface between the CPU and the 82588 is totally eliminated.

7.5.5 iSBX Interface to StarLAN

Figure 7-37 shows how to interface the 82588 in a StarLAN environment to the iSBX bus. It uses 2 DMA channels—tapping the second DMA channel from a neighboring iSBX connector. Such a board can be used to make a StarLAN to an Ethernet or a SNA or DECNET gateway when it is placed on an appropriate SBC board. It may also be used to give a StarLAN access to any SBC board (with an iSBX connector) independent of the type of processor on the board.

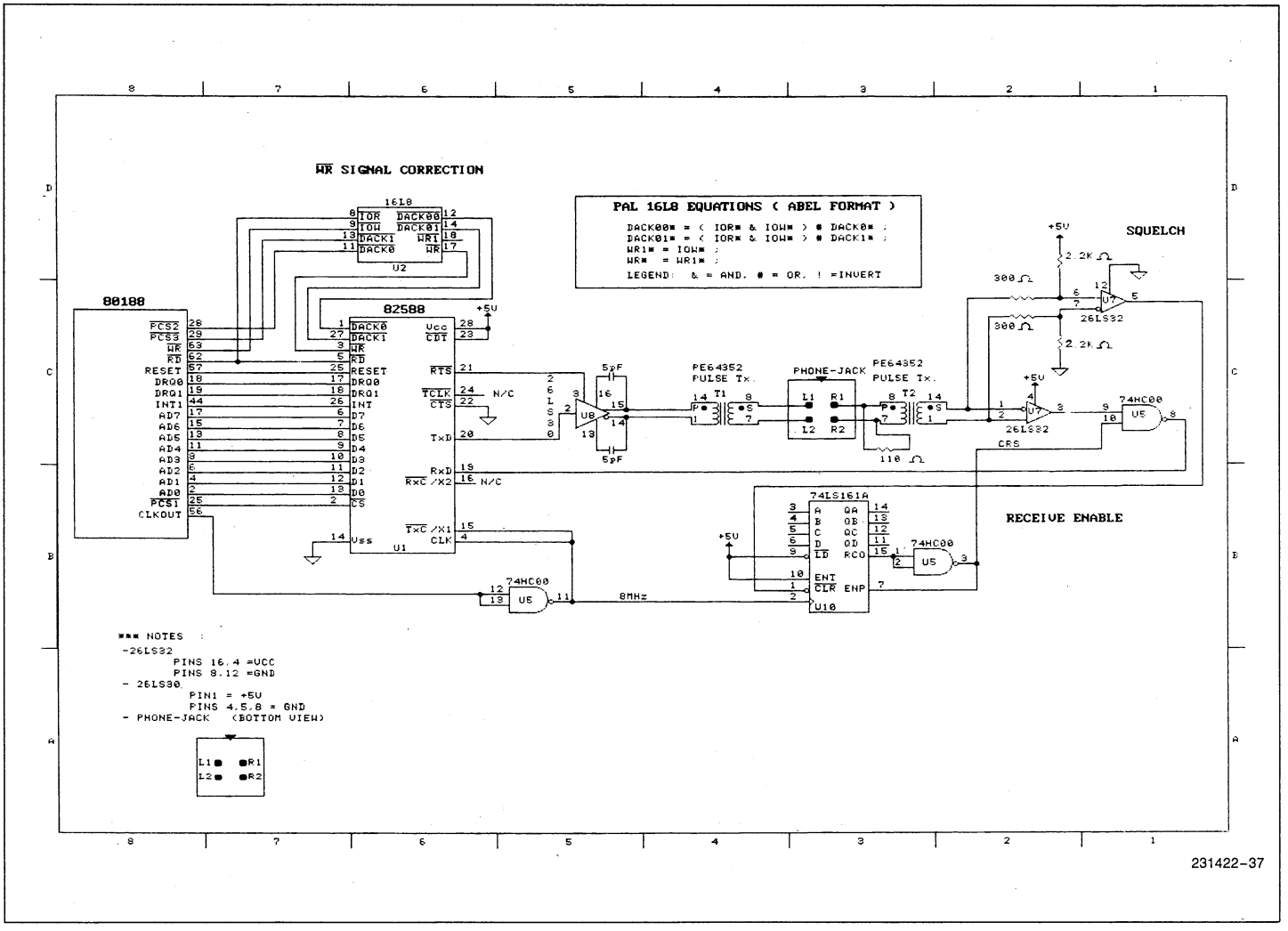


Figure 7-36. 80188 Interface to 82588

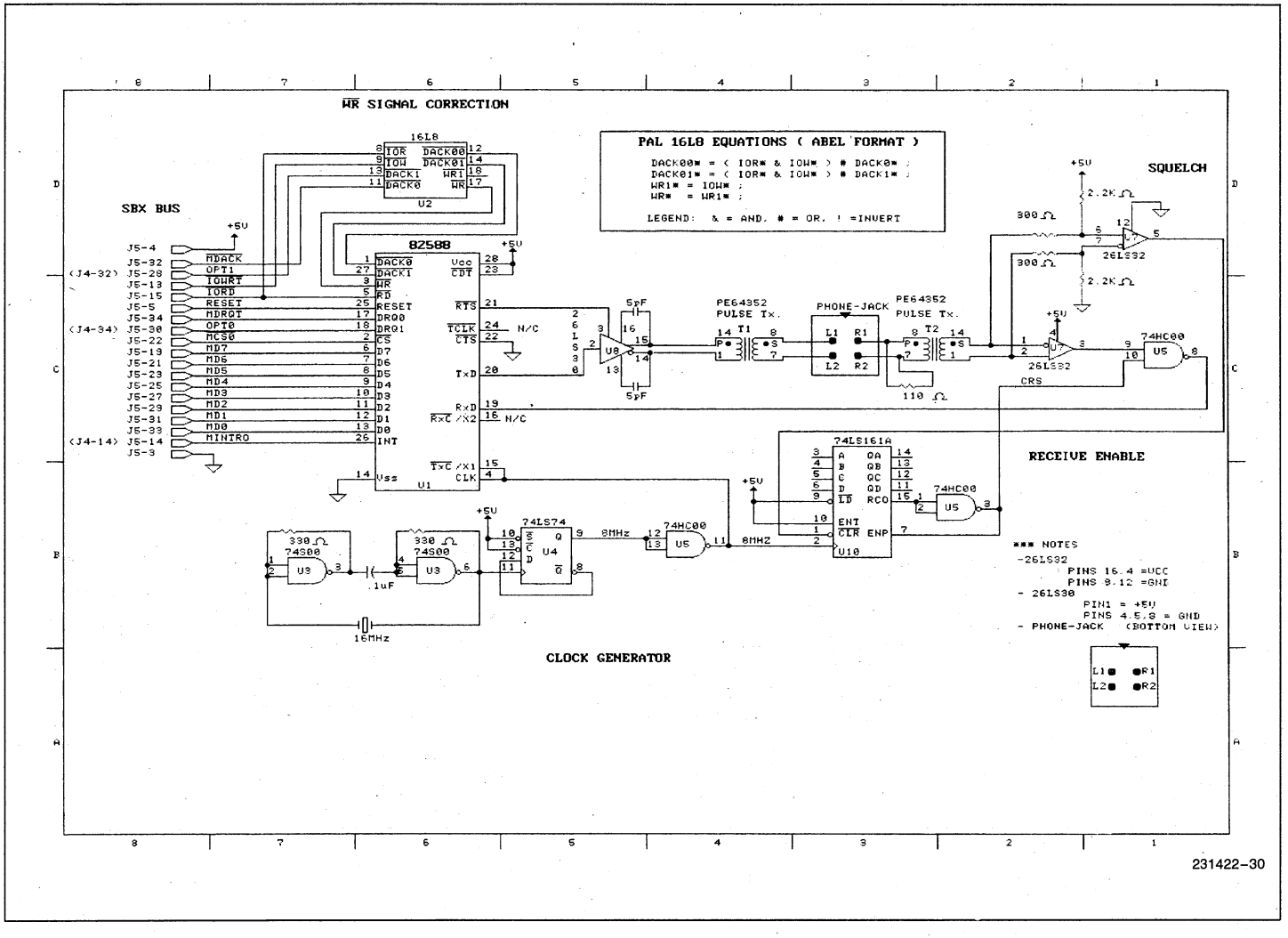


Figure 7-37. iSBX Interface to 82588

7.5.6 Protection Circuits

Protection from high voltage on the cable can be achieved by connecting zener diodes to the pulse transformers as shown in Figure 7-38. The pulse transformers also protects the node from up to 2000 volts on the link.

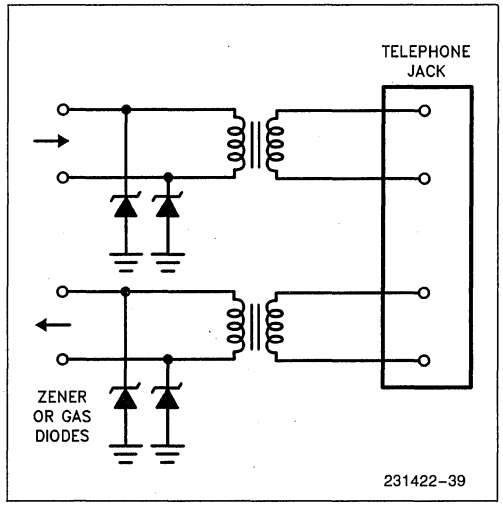


Figure 7-38. Protection Circuits

7.6 THE StarLAN HUB

The function of a StarLAN HUB is described in section 7.2. Figure 7-39 shows a block diagram of a HUB. It receives signals from the nodes (or lower level HUBs) detects if there is a collision, generates the collision presence signal, re-times the signal and sends it out to the higher level HUB. It also receives signal from the higher level HUB, re-times it and sends it to all the nodes and lower level HUBs connected to it. If there is no higher level HUB, a switch on the HUB routes the upstream received signal down to all the lower nodes as shown in Figure 7-39. The functions performed by a HUB are:

*Receiving signals, squelch
*Carrier Sensing
*Collision Detection
*Collision Presence Signal Generation
*Signal Retiming
*Driving signals on to the cable
*Jabber Function

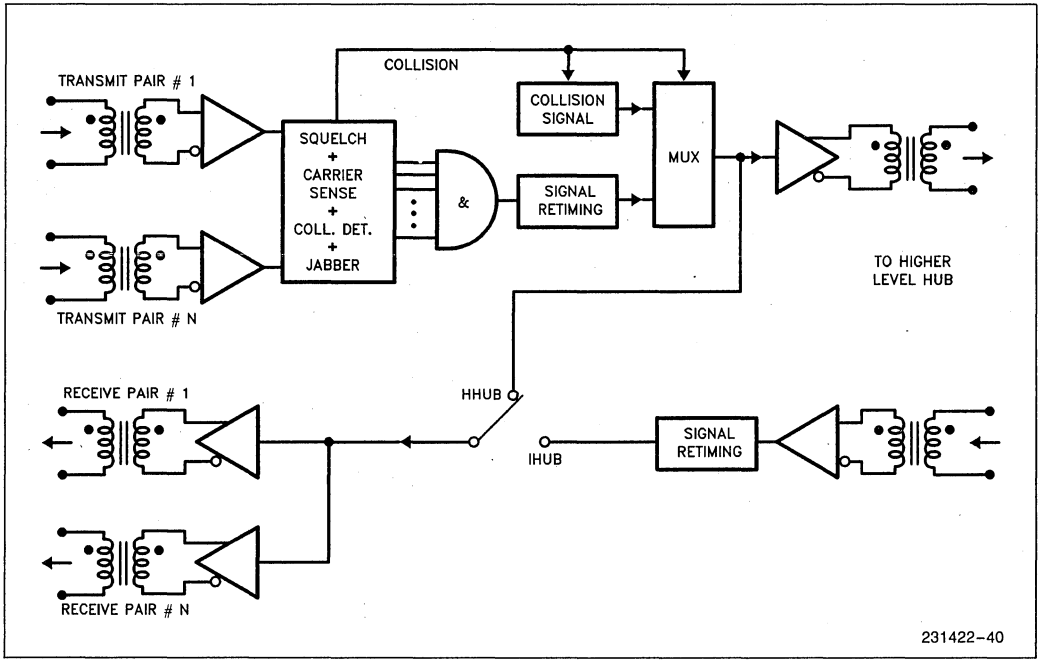


Figure 7-39. StarLAN HUB

7.6.1 The StarLAN HUB Design

Figure 7-40 shows the implementation of a 4 node, 1 level HUB. It is a header HUB with 4 ports. It performs all of the above mentioned functions except for the signal re-timing—which is not essential for a 1 level HUB, especially with 82588 based nodes which can tolerate a considerable amount (up to 120 ns) of jitter. Using the circuit in Figure 7-40, the design of the HUB will now be explained.

7.6.1.1 THE RECEIVING CIRCUITS AND CARRIER SENSING

The transmitted signal from each node or from a lower level HUB are received by the HUB through a line termination resistor of 110Ω and an isolation pulse transformer. The circuit, as seen in the upper right hand corner of the Figure 7-40, is identical to the receive circuit on the StarLAN board in Figure 7-27. Refer to section 7.5.2.2 for the description of the squelch and frame envelope detection circuits. The output of the envelope detection circuit is the Enable signal which is active whenever there is activity on the channel. From each of the input channels to the HUB we get one received signal, Rn and an enable (or Carrier Sense) signal En.

7.6.1.2 COLLISION DETECTION

Rn and En signals from each channel are fed to a 16L8 PAL. The PAL contains the logic for the following functions:

*Collision Detection

*Output Signal Selection

*Enabling the RS-422 Drivers

Collision Detection in the StarLAN HUB is performed by detecting the presence of activity on more than one input channel. This means if the signal En is active for more than one channel, a collision is said to occur. This translates to the PAL equation:

```
COLLIS = ENABL &
         !(( EA & !EB & !EC & !ED) #
           (!EA & EB & !EC & !ED) #
           (!EA & !EB & EC & !ED) #
           (!EA & !EB & !EC & ED));
```

where

```
ENABL = EA # EB # EC # ED;
```

ENABL is active whenever at least one input channel is active and its complement is used to turn on the RS-422 drivers.

COLL is the complement of COLLIS, and is used to set the Collision flip-flop. This flip-flop remains set till the ENABL signal goes inactive again—till activity on all input channels have died out. The output of the Collision flip-flop, COLLEN, goes to the select input of the multiplexor (shown in Figure 7-39) which selects between the input signal (RCV)—in case of no collision—and the Collision Presence Signal (CS)—in case of collision. The multiplexor is also implemented in the PAL using the equation:

```
SIGNAL = (RCV & !COLLEN) # (CS & COLLEN);
```

where RCV is a qualified received signal—each input signal Rn qualified by the respective enable signal En—and also incorporating the AND gate as shown in the Figure 7-39. The AND gate has the function of selecting the active signal. Since the idle state of the signals is high, the single active signal is selected out by an AND function of all the input signals.

```
RCV = (RA # !EA) & (RB # !EB) & (RC # !EC) & (RD # !ED);
```

The 16L8 PAL has thus been used to perform the functions of qualifying the received signal, selecting the active signal, enabling the output drivers, detecting a collision and multiplexing the output signal between the received signal and the Collision Presence Signal.

7.6.1.3 THE COLLISION PRESENCE SIGNAL

The Collision Presence Signal (CPS) is generated by the HUB whenever the HUB detects a collision. It then propagates the CPS to the higher level HUB. The CPS signal pattern is shown in Figure 7-41. Whenever a StarLAN node receives this signal, it should be able to detect within a very few bit times that a collision occurred. Since the nodes detect the occurrence of a collision by detecting violations in Manchester encoding, the CPS must obviously be a signal which violates Manchester encoding. Figure 7-15 shows that the CPS has missing mid-cell transitions occurring every two and a half bit cells. These are detected as Manchester code violations. Thus, the StarLAN node is presented with collision detection indications every two and a half microseconds. This results in fast and reliable detection of collisions. CPS has a period of 5 microseconds.

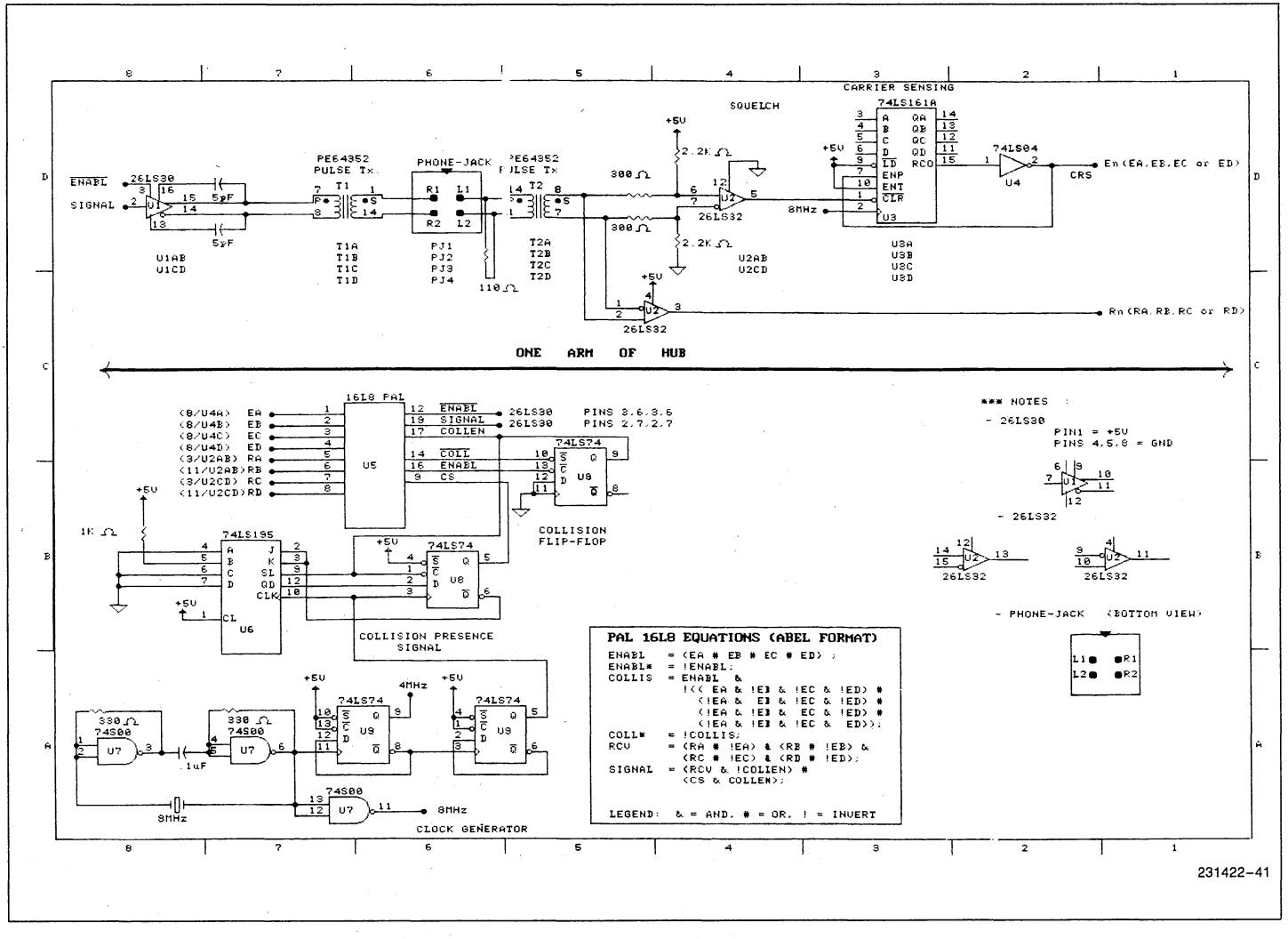
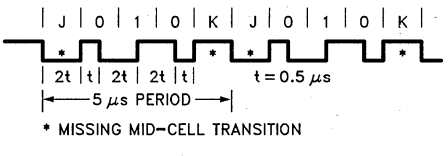


Figure 7-40. StarLAN HUB Schematics



- Collision Presence Signal (CPS) is generated by the HUB when it detects more than one input line active.
- CPS violates Manchester encoding rules—due to missing mid-cell transitions—hence is detected as a collision by the DTE (82588).

Choice of Collision Presence Signal

- It is a Manchester look-alike signal—edges are 0.5 or 1.0 μs apart.
  —Identical radiation, crosstalk and jitter characteristics
  —Eases retiming of the signal in the HUB
- It is easy to generate—1.5 TTL pack, or in a PAL

Figure 7-41. Collision Presence Signal

One may wonder why such a strange looking signal was selected for CPS. The rationale is that this CPS looks very much like a valid Manchester signal—edges are 0.5 or 1.0 microsec. apart—resulting in identical radiation, cross-talk and jitter characteristics as a true Manchester. This also makes the re-timing logic for the signals simpler—it need not distinguish between valid Manchester and CPS. Moreover, this signal is easy to generate.

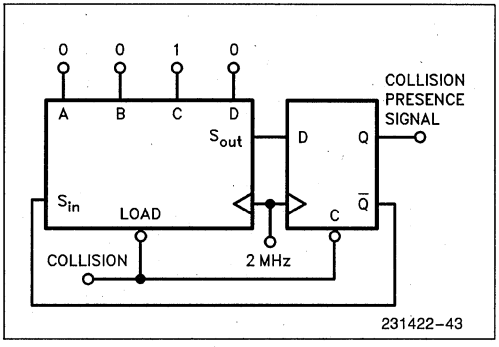


Figure 7-42. Collision Presence Signal Generation

Two important requirements for CPS are: a) it should be generated starting with a low phase and b) once it starts, it should continue until *all* the input lines to the HUB die out. Typically, when the collision occurs, the multiplexor in the HUB switches from RCV signal to the CPS. If just before switching the phase of the RCV signal is high and if CPS were to start with a high, the output signal going back to the nodes may remain high for over 1.5 bit times. This would be interpreted by the node, according to IEEE 802.3 specifications, as a loss of Carrier. The restriction a) prevents this. The restriction b) ensures that the CPS is seen by all nodes on the network since it is generated until every node has finished generating the Jam pattern.

CPS is generated using a 4 bit shift register and a flip-flop as shown in Figure 7-42. It works off a 2 MHz clock. A closer look at the CPS waveform shows that it is inverse symmetric within the 5 microseconds period. The circuit is a 5 bit shift register with a complementary feedback from the last to the first bit. The bits remain in defined states (01100) till collision occurs. On collision the bits start rotating around generating the pattern of 0011011001, 0011011001, 00110 . . . with each state lasting for 0.5 microseconds.

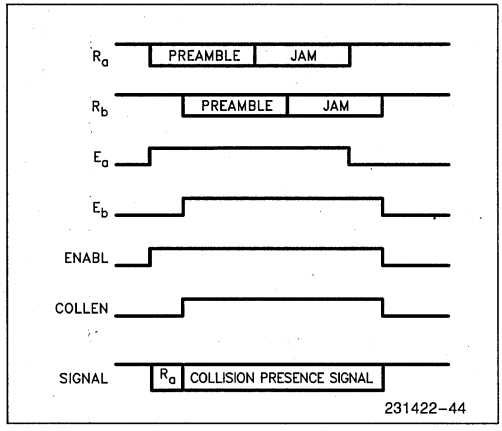


Figure 7-43. Collision Scenario at the HUB

Figure 7-43 shows a typical collision scenario at the HUB. Two nodes A and B with their signal Ra and Rb collide. Ea and Eb are their carrier sense or enable signals. The output SIGNAL could be seen switching

from Ra to the Collision Presence Signal as soon as Ea and Eb are both active. CPS remains active till COLLEN remains active—i.e. till either Ea or Eb is active.

7.6.1.4 SIGNAL RETIMING

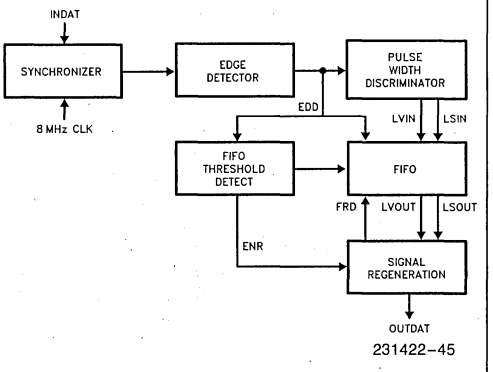
Whenever the signal goes over a cable it suffers jitter. This means that the edges are no longer separated by the same 0.5 or 1.0 microseconds as at the point of origin. There are various causes of jitter. Drivers, receivers introduce some shifting of edges because of differing rise and fall times and thresholds. A random sequence of bits also produces a jitter. A maximum of 90 ns of jitter can accumulate in a StarLAN network from a node to a HUB or from a HUB to another HUB. The following values are proposals and are not yet finalized in the 1BASE5 standards draft (June 1985):

Transmitter	±5 ns peak
Cable Intersymbol	±20 ns peak
Cable Interference	±50 ns peak
Receiver	±5 ns peak
HUB	±10 ns peak
Total	±90 ns peak

It is important that the signal is cleaned up of this jitter before it is sent on the next stretch of cable because if too much jitter accumulates, the signal is no longer meaningful. A valid Manchester signal would, as a result of jitter, may no longer look like valid Manchester. The process of either re-aligning the edges or reconstructing the signal or even re-generating the signal so that it once again "looks new" is called re-timing. Its also called "dejittering". StarLAN requires that the signal is re-timed after it has travelled on a segment of cable. In a typical HUB two re-timing circuits are necessary; one for the signals going upstream towards the higher level HUB and the other for signals going downstream towards the nodes.

7.6.1.5 DESIGNING THE RETIMING CIRCUIT

The HUB shown in Figure 7-40 does not have a re-timing circuit. However, this section will discuss the principles of designing a re-timing circuit. Figure 7-44 shows the block diagram of a re-timing circuit. The data coming in is synchronized using an 8 MHz sampling clock. Edges in the waveform are detected doing an XOR of two consecutive samples. A counter counts the number of 8 MHz clocks between two edges. This gives an indication of long (6 to 10 clocks) or short (3 to 5 clocks) pulses in the received waveform. Pulses shorter than 3 clocks and longer than 10 clocks are ignored—allowed to pass through. It is assumed that these conditions occur only during idle state. Every time an edge occurs, the polarity of the waveform and the length—(S)hort or (L)ong—of the pulse is fed into the FIFO. Retiming of the waveform is done by actually generating a new waveform based on the information being pumped into the FIFO. The signal regeneration unit reads the FIFO and generates the output waveform out of 8 MHz clock pulses based on what it reads:



SIGNALS DEFINITION

indat	—	Input data
edd	—	edge detection pulse, used to load output of pulse descriminator in the fifo, and to increment the threshold detector.
lvin	—	pulse logic level, input to fifo.
lsin	—	pulse long/short descrimination, input to fifo
frd	—	fifo read pulse
lvout	—	pulse logic level, output from fifo.
lsout	—	pulse long/short descrimination, output from fifo.
enr	—	enable pulse regeneration, a function of fifo threshold.

Figure 7-44. Signal Retiming Circuit

FIFO	Output
S,1	1111
S,0	0000
L,0	00000000
L,1	11111111

Example:

Input Waveform . . .

0011110000000111111111100011111100

Input into the FIFO: <S,1> <L,0> <S,0> <L,1> <S,1>

<S,1>,<L,0>,<L,1>,<S,0>,<S,1> from the FIFO is regenerated as:
1111000000001111111100001111

It could be seen that the output always has edges separated by 4 or 8 clock pulses—0.5 or 1.0 microseconds.

The FIFO is primarily needed to account for a difference of clock frequencies at the source and regeneration end. Due to this difference, data can come in faster or slower than the regeneration circuit expects. A 16 deep FIFO can handle frequency deviations of up to 200 ppm for frame lengths up to 1600 bytes. The FIFO also overcomes short term variations in edge separation. It is essential that the FIFO fills in up to about half before the process of regeneration is started. Thus, if the regeneration is done at a clock slightly faster than the source clock, there is always data in the FIFO to work from. That is why the FIFO threshold detect logic is necessary, which counts 8 edges and then enables the signal regeneration logic.

7.6.1.6 DRIVER CIRCUITS

The signal coming out of the PAL is sent back to the nodes in a 1 level HUB. The driver circuit used is identical to the one used in the node on the StarLAN board. Am26LS30 RS-422 driver is used to drive the pulse transformer. The slew rate capacitors on the drivers increase the rise and fall times of the pulses to 150 ns as required by StarLAN to overcome the cross-talk and radiation problems. The same signal is sent to all nodes on different drivers, pulse transformers and wires. For a multi-level HUB, the routing of signals is done as shown in Figure 7-39.

7.6.1.7 JABBER FUNCTION

This design does not implement the jabber unit but it is described here for completeness. IEEE 802.3 does not require this feature; it is an option. The jabber function in the HUB is to deal with abnormally long transmissions on the network by any node. The jabber unit monitors the time taken by any single transmission. If this exceeds a time-out value T1, then the HUB transmits the CPS signal until all inputs become idle. If all inputs are not idle in time T2, then the Jabber unit disables (or ignores) the active inputs and treats them as idle. The Jabber unit can re-enable the disabled inputs after a time T3. These timing relations are shown in Figure 7-45. It shows the outputs JT1, JT2 and JT3 of the 3 timers needed to implement this function. Instances (1), (2), (3) and (4) show the following events and actions:

(1)—Start of transmission.

(2)— Jabber Timer 1 times out here, if the input(s) are active, Timer 2 is started and CPS is generated and propagated.

(3)— Timer 2 runs out. CPS is stopped. If input(s) not yet idle, the active inputs are disabled. Timer 3 is started.

(4)— Timer 3 runs out. Disabled units may be enabled.

The current definitions of the jabber timers T1, T2 and T3 are:

T1: 25–100 ms; 2–8 times maximum frame size

T2: 5–40 ms; 10–80 times slot time

T3: 20–80 times T1

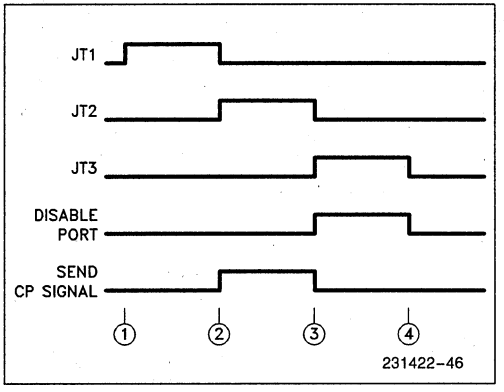


Figure 7-45. Jabber Timing Relations

7.6.2 HUB Reliability

Since the StarLAN HUBs form focal points in the network, it is obviously important that they are very reliable, since it can be single point of failure which can affect a number of nodes or can even bring down the whole network. Initial studies done by AT&T on their 20 node HUB have shown that they have a MTBF of 7 years and the most unreliable part is the connector (the telephone jack). Additional studies done by TANDEM Computers have shown that a fault tolerant HUB is not a necessity.

7.7 SOFTWARE DRIVER

The software needed to drive the 82588 in a StarLAN environment is not different from that needed in a generic CSMA/CD environment. This section goes into specific procedures used for operations like TRANSMIT, RECEIVE, CONFIGURE, DUMP, ADDRESS SET-UP, etc. A special treatment will be given to interfacing with the IBM PC—DMA, interrupt and I/O. Since all the routines were written and tried out in PLM-86 and ASM-86, all illustrations are in these languages.

7.7.1 Interfacing to IBM PC

The StarLAN board interfaces to the CPU, DMA controller and the interrupt controller on the IBM PC system board. The software to operate the 82588 runs on the system board CPU. The illustrated routines in this section show exactly how the software interface works between the system resources on the IBM PC and the StarLAN board.

7.7.1.1 DOING I/O ON IBM PC

The safest way to use the PC monitor as an output device and the keyboard as the input device is to use them through DOS system calls. The following is a set of routines which are handy to do most of the I/O:

ks — to find out if a new key has been pressed

ci — to read a key from the keyboard

co — to display a character on the screen

cos— to display a character string on the screen

cis — to read in a character string from the keyboard

The exact semantics and the protocol for doing these functions through DOS system calls is shown in the listing in Figure 7-46. Refer to the DOS Manual for a more detailed description. To make a DOS system call, register AH of 8088 is loaded with the call Function Number and then, a software interrupt (or trap) 21 hex is executed. Other 8088 registers are used to transfer any parameters between DOS and the calling program. The code is written in Assembly language for register access. Let us take an example of the 'cos' routine:

```
lds dx,STRING__POINTER ; load pointer to string in
                         reg. ds:dx
mov ah,09h             ; 9 = function number
                         for string o/p
int 21h                ; DOS System Call
```

These procedures are called from another module, written in a higher level language like PLM-86. The parameters are transferred to the ASM-86 routines on the stack.

Examples of using the I/O routines:

```
KEY__STATUS = ks;
                    /* inquire keyboard status */
NEW__KEY = ci;
                    /* input new key */
call cis(@LINE__BUFFER);
                    /* string input */
call co(CHAR__OUT);
                    /* to output CHAR__OUT on screen */
call cos(@('THIS IS A MESSAGE.$'));
                    /* output string */
                    /* note $ terminator */
```

7.7.2 Initialization and Declarations

Figure 7-47 shows some declarations describing what addresses the devices have and also some literals to help understand the other routines in this section.

Figure 7-48 shows the initialization routines for the IBM PC and for the 82588. It also shows some of the typical values taken by the memory buffers for Configure, IA__Set, Multicast and transmit buffers.

7.7.3 General Commands

Operations like Transmit, Receive, Configure etc. are done by a simple sequence of loading the DMA controller with the necessary parameters and then writing the command to the 82588.

Example: Configure Command

To configure the operating environment of the 82588. This command must be the first one to be executed after a RESET.

```
call DMA__LOAD(1,1,12,@CONFIG__588);
output (CS__588) = 12h;
```

The first statement is the prologue to the configure command to the 82588 which calls a routine to load and initialize the DMA controller for the desired operation. this routine is described in section 7.7.4. The parameters for DMA__LOAD are:

first parameter = 82588 channel number (= 1)
second parameter = direction (= 1, memory to 82588)
third parameter = length of DMA transfer (= 12)
fourth parameter = pointer to memory buffer (= @CONFIG__588)

The second statement writes 12h to the command register of the 82588 to execute a Configure command on channel 1.

When the command execution is complete (successfully or not), 82588 interrupts the 8088 CPU through the 8259A, on the system board. This executes the interrupt service routine, described in section 7.7.5, which takes the epilogue action for the command.

Most operations are very similar in structure to Configure. The 82588 Reference Manual describes them in detail. Figure 7-49 shows a listing of the most commonly used operations like:

CONFIGURE	INDIVIDUAL-ADDRESS SET-UP	(IA)
TRANSMIT	MULTICAST-ADDRESS SET-UP	(MC)
DIAGNOSE	RECEIVE (RCV)-ENABLE	
DUMP	RECEIVE (RCV)-DISABLE	
TDR	RECEIVE (RCV)-STOP	
RETRANSMIT	READ-STATUS	

7.7.4 DMA Routines

DMA_LOAD procedure is used to program the 8237A DMA controller for all the operations requiring DMA service. It also starts or enables the programmed DMA channel after programming it. Figure 7-50 shows the listing of this procedure. It accepts 4 parameters from the calling routine to decide the programming configuration for the 8237A. The parameters for DMA_LOAD are C, D, L and P:

first parameter - C - = 82588 Channel number
second parameter- D - = Direction
third parameter - L - = Length of DMA transfer
fourth parameter - P - = Pointer to memory buffer

if P = 0 then 8237A channel = 1;
if P = 1 then 8237A channel = 3;

if D = 0 then transfer is from 82588 to memory block
if D = 1 then transfer is from memory block to 82588

L > = data block to be transferred

Note that L need not be exactly equal to the length of the block of data to be transferred. The 82588 stops generating DMA requests after it has performed the required number of data transfers.

P is in segment:offset form. The first part of the DMA_LOAD procedure converts this to a linear 20 bit form. The lower 16 bits are loaded into the DMA Address register (dma_addr) of 8237A in two 8 bit write operations. The upper 4 bits are loaded into the Page register (dma_addrh). Note that there is no overflow from the 8237A address register to the page register.

Figure 7-51 is a listing of the DMA_LOAD procedure for the 80188 or 80188 on-chip DMA controller. It has the same caller interface as the 8237A based one.

7.7.5 Interrupt Routine

The interrupt service routine, 'intr_588', shown in Figure 7-52, is invoked whenever the 82588 interrupts. It is basically a reentrant interrupt procedure that starts with re-enabling the interrupts—to permit a receive interrupt preempt the post transmit interrupt processing. It takes action based on what event has caused the interrupt. For all the events that use DMA, it disables the DMA channel. For the transmit and retransmit events, it increments some statistical data counters based on the status information. For the receive event, it first of all extracts the receive status information at the end of the received frame—even in case of a multiple buffer reception—and increments statistical data counters. This is a dummy interrupt handler. A real interrupt handler would do functions like buffer release, buffer acquisition, etc.

Interrupt service routines should be kept as short as possible. This is to enable reception of back-to-back frames and also to transmit frames separated by interframe spacing.

```
$title('..... I/O Routines for the IBM PC .....')

; Sharad Gandhi, DCO Technical Marketing, INTEL Corp.

; Routines to do I/O on the IBM PC

;=============================================================
;            Declarations in the Calling PLM-86 Routine
;=============================================================
;
;ks: procedure byte external;        /* key status routine */
;end ks;
;
;ci: procedure byte external;        /* console input routine */
;end ci;
;
;co: procedure(char) external;       /* console output routine */
;declare char byte;
;end co;
;
;cos: procedure(str_ptr) external;  /* console string output routine */
;declare str_ptr pointer;
;end cos;
;
;cis: procedure(str_ptr) external;  /* console string input routine */
;declare str_ptr pointer;
;end cis;
;
;=============================================================

  name  pcio

  public  ks,ci,co,cos,cis

stal  struc       ;stack layout
old_bpl dw  ?
old_ipl dw  ?
str_ptr dd  ?
stal  ends

sta2  struc       ;stack layout
old_bp2 dw  ?
old_ip2 dw  ?
char    db  ?
sta2  ends

  cgroup  group code

code  segment public  'code'
```

Figure 7-46. I/O Drivers for IBM PC

```
  assume  cs:cgroup

;--------------Keyboard Status----------------

ks  proc  near

  mov   ah,0bh  ; to check key input status
  int   21h     ; DOS function call
  ret           ; key status in AL register

ks  endp

;--------------Console Input-----------------

ci  proc  near

  mov   ah,08h  ; to get key input from PC
  int   21h     ; DOS function call
  ret           ; key in AL register

ci  endp

;--------------Console Output----------------

co  proc  near

  push  ax
  push  dx
  mov   dl,[bp].char; character from stack
  mov   ah,02h  ; output character to PC
  int   21h     ; DOS function call
  pop   dx
  pop   ax
  ret   2

co  endp
```

Figure 7-46. I/O Drivers for IBM PC (Continued)

```
;-------------Console String Output-------------

cos proc  near

  push  bp
  mov   bp,sp
  push  ds
  push  dx
  push  ax

  lds   dx,[bp].str_ptr
  mov   ah,09h  ; output character string to PC
  int   21h     ; DOS function call

  pop   ax
  pop   dx
  pop   ds
  pop   bp
  ret   4

cos endp

;-------------Console String Input-------------

cis proc  near

  push  bp
  mov   bp,sp
  push  ds
  push  dx
  push  ax

  lds   dx,[bp].str_ptr
  mov   ah,0ah  ; input character string from PC
  int   21h     ; DOS function call

  pop   ax
  pop   dx
  pop   ds
  pop   bp
  ret   4

cis endp

;-----------------------------------------------

code  ends

  end
```

Figure 7-46. I/O Drivers for IBM PC (Continued)

```
/*-------------------------------------------------------------------------*/

/* chip address declarations */

  declare cs_588   literally '0300h';   /* 82588 command/status */

  declare pic_mask literally '021h';    /* 8259A interrupt controller */
  declare pic_ocw2 literally '020h';

  declare dma_req  literally '0ah';     /* 8237A DMA Controller */
  declare dma_mode literally '0bh';
  declare dma_flff literally '0ch';

  declare dma_addr_1 literally   '02h';
  declare dma_bc_1 literally     '03h';
  declare dma_addrh_1 literally '080h';

  declare dma_addr_3 literally   '06h';
  declare dma_bc_3 literally     '07h';
  declare dma_addrh_3 literally '082h';

/*-------------------------------------------------------------------------*/

/* literals */

  declare dma_on_1 literally '01h';
  declare dma_on_3 literally '03h';
  declare dma_off_1 literally '05h';
  declare dma_off_3 literally '07h';


  declare enable_588 literally '11011111b'; /* unmask level 5 */
  declare seoi_pico literally  '01100101b'; /* specific EOI level 101 */

  declare dma_rx_mode_1   literally '01000101b'; /* rx channel # 1 */
       /* single byte, rx mode, channel 1 */

  declare dma_rx_mode_3   literally '01000111b'; /* rx channel # 3 */
       /* single byte, rx mode, channel 3 */

  declare dma_tx_mode_1   literally '01001001b'; /* tx channel # 1 */
       /* single byte, tx mode, channel 1 */

  declare dma_tx_mode_3   literally '01001011b'; /* tx channel # 3 */
       /* single byte, tx mode, channel 3 */

/*-------------------------------------------------------------------------*/
```

Figure 7-47. Literal Declarations

```
/*-------------------------------------------------------------------------*/

/* system initialize */

sys_init: procedure;

  call set$interrupt (13,intr_588); /* base 8, level 5 */
  output(pic_mask) = input(pic_mask) and enable_588;
  output(pic_ocw2) = seoi_pico;
  intr_588_flag = 0;

end sys_init;

/*-------------------------------------------------------------------------*/

/* 82588 init */

init_588: procedure;

  config_588(00) = 10;          /* to configure all 10 parameters */
  config_588(01) = 00;
  config_588(02) = 00001000b;   /* mode 0, 8  MHz clock, 1 Mb/s */
  config_588(03) = buff_len/4;  /* Receive Buffer length */
  config_588(04) = 00100110b;   /* No loopback, addr len = 6, Preamble = 8 */
  config_588(05) = 00000000b;   /* Differential Manchester = off  */
  config_588(06) = 96;          /* IFS = 96 TCLK */
  config_588(07) = 0;           /* Slot time = 512 TCLK */
  config_588(08) = 11110010b;   /* Max. No. Retries = 15 */
  config_588(09) = 00000100b;   /* Manchester encoding */
  config_588(10) = 10001100b;   /* Internal CRS and CDT, CRSF = 4 */
  config_588(11) = 64;          /* Min frame length = 64 bytes = 512 bits */
```

Figure 7-48. Initialization Routines

```
   ia_set_buff_588(0) = 6;
   ia_set_buff_588(1) = 0;
   ia_set_buff_588(2) = 000h;
   ia_set_buff_588(3) = 041h;
   ia_set_buff_588(4) = 000h;
   ia_set_buff_588(5) = 000h;
   ia_set_buff_588(6) = 000h;
   ia_set_buff_588(7) = 000h;

   multicast_buff_588(00) = 12;
   multicast_buff_588(01) = 00h;
   multicast_buff_588(02) = 11h;
   multicast_buff_588(03) = 12h;
   multicast_buff_588(04) = 13h;
   multicast_buff_588(05) = 14h;
   multicast_buff_588(06) = 15h;
   multicast_buff_588(07) = 16h;
   multicast_buff_588(08) = 21h;
   multicast_buff_588(09) = 22h;
   multicast_buff_588(10) = 23h;
   multicast_buff_588(11) = 24h;
   multicast_buff_588(12) = 25h;
   multicast_buff_588(13) = 26h;

   tx_buffer_588(00) = tx_frame_len mod 256;
   tx_buffer_588(01) = tx_frame_len / 256;
   tx_buffer_588(02) = 011h;   /* initial destination address = MC(1) */
   tx_buffer_588(03) = 012h;
   tx_buffer_588(04) = 013h;
   tx_buffer_588(05) = 014h;
   tx_buffer_588(06) = 015h;
   tx_buffer_588(07) = 016h;

end init_588;

/*-------------------------------------------------------------------------------*/
```

Figure 7-48. Initialization Routines (Continued)

```
/*-------------------------------------------------------------------------*/

ia_set: procedure;              /* command - 01 */

  call dma_load(1,1,8,@ia_set_buff_588);
  intr_588_flag = 0ffh;
  output (cs_588) = 11h;        /* ia_set to channel 1 */
  return;

end ia_set;

/*-------------------------------------------------------------------------*/

config: procedure;              /* command - 02 */

  call dma_load(1,1,12,@config_588);
  intr_588_flag = 0ffh;
  output (cs_588) = 12h ;       /* configure to channel 1 */
  return;

end config;

/*-------------------------------------------------------------------------*/

multicast: procedure;           /* command - 03 */

  call dma_load(1,1,14,@multicast_buff_588);
  intr_588_flag = 0ffh;
  output (cs_588) = 13h;        /* multicast to channel 1 */
  return;

end multicast;

/*-------------------------------------------------------------------------*/

transmit: procedure(buffer_len,buffer_pointer); /* command - 04 */

  declare buffer_len word;
  declare buffer_pointer pointer;

  tx_buffer_588(00) = buffer_len mod 256;
  tx_buffer_588(01) = buffer_len / 256;
  tx_buff_ptr = buffer_pointer;
  call dma_load(1,1,1536,tx_buff_ptr);
  intr_588_flag = 0ffh;
  output (cs_588) = 14h;        /* transmit to channel 1 */
  return;

end transmit;

/*-------------------------------------------------------------------------*/
```

Figure 7-49. General Commands

```
/*-------------------------------------------------------------------------*/

tdr: procedure;                 /* command - 05 */

  intr_588_flag = 0ffh;
  output (cs_588) = 5;          /* tdr command */
  return;

end tdr;

/*-------------------------------------------------------------------------*/

dump_588: procedure;            /* command - 06 */

  call dma_load(1,0,64,@dump_buff_588);
  intr_588_flag = 0ffh;
  output (cs_588) = 16h;        /* dump to channel 1 */
  return;

end dump_588;

/*-------------------------------------------------------------------------*/

diagnose: procedure;            /* command - 07 */

  intr_588_flag = 0ffh;
  output (cs_588) = 7;          /* diagnose command */
  return;

end diagnose;

/*-------------------------------------------------------------------------*/

rcv_enable: procedure(channel,buffer_ptr); /* command - 08 */

  declare channel byte;
  declare buffer_ptr pointer;

  buff_alloc = 1;               /* # of buffers allocated */
  call dma_load(channel,0,1536,buffer_ptr);
  output(cs_588)= 8;
  return;

end rcv_enable;

/*-------------------------------------------------------------------------*/
```

Figure 7-49. General Commands (Continued)

```
/*-------------------------------------------------------------------------*/

rcv_disable: procedure;        /* command - 10 */

  output(cs_588)= 10;
  return;

end rcv_disable;

/*-------------------------------------------------------------------------*/

rcv_stop: procedure;           /* command - 11 */

  output(cs_588)= 11;
  return;

end rcv_stop;

/*-------------------------------------------------------------------------*/

retransmit: procedure;         /* command - 12 */

  call dma_load(1,1,1536,tx_buff_ptr);
  intr_588_flag = 0ffh;
  output (cs_588) = 1ch;       /* retransmit to channel 1 */
  return;

end retransmit;

/*-------------------------------------------------------------------------*/

read_status_588: procedure;    /* command - 15 */

  output (cs_588) = 15;        /* release pointer, initial = 00 */

  status_588(0) = input (cs_588);
  status_588(1) = input (cs_588);
  status_588(2) = input (cs_588);
  status_588(3) = input (cs_588);
  return;

end read_status_588;

/*-------------------------------------------------------------------------*/
```

Figure 7-49. General Commands (Continued)

```
/*-------------------------------------------------------------------------*/

dma_load: procedure(channel,direction,trans_len,buff_ptr) reentrant;

  declare channel byte;   /* channel # */
  declare direction byte;   /* 0 = rx, 588 -> mem; 1 = tx, mem -> 588 */
  declare trans_len word;   /* byte count */
  declare buff_ptr pointer; /* buffer pointer in seg:offset form */

  declare buff_ptr_20bit dword; /* convert buff_ptr to 20bit buff_ptr */
  declare ptr1 pointer;
  declare (wrd based ptr1)(2) word;
  ptr1 = @buff_ptr;       /* wrd (0,1) overlaps buff_ptr */
  buff_ptr_20bit = shl((buff_ptr_20bit := wrd(1)),4) + wrd(0);
  ptr1 = @buff_ptr_20bit;     /* wrd (0,1) overlaps buff_ptr_20bit */

  do case channel and 00000001b;
    do case direction and 00000001b;
      do;             /* channel # 1 , 588(0) to memory */
      output(dma_req)     = dma_off_1;
      output(dma_flff)    = 0;    /* clear first/last flip-flop */
      output(dma_mode)    = dma_rx_mode_1;
      output(dma_addr_1)  = low (wrd(0));
      output(dma_addr_1)  = high(wrd(0));
      output(dma_addrh_1) = low (wrd(1));
      output(dma_bc_1)    = low (trans_len);
      output(dma_bc_1)    = high(trans_len);
      output(dma_req)     = dma_on_1; /* start DMA channel # 1 */
      end;

      do;             /* channel # 1 , memory to 588(0) */
      output(dma_req)     = dma_off_1;
      output(dma_flff)    = 0;    /* clear first/last flip-flop */
      output(dma_mode)    = dma_tx_mode_1;
      output(dma_addr_1)  = low (wrd(0));
      output(dma_addr_1)  = high(wrd(0));
      output(dma_addrh_1) = low (wrd(1));
      output(dma_bc_1)    = low (trans_len);
      output(dma_bc_1)    = high(trans_len);
      output(dma_req)     = dma_on_1; /* start DMA channel # 1 */
      end;

    end;
```

Figure 7-50. DMA Routine

```
    do case direction and 00000001b;

      do;             /* channel # 3 , 588(1) to memory */
      output(dma_req)     = dma_off_3;
      output(dma_flff)    = 0;    /* clear first/last flip-flop */
      output(dma_mode)    = dma_rx_mode_3;
      output(dma_addr_3)  = low (wrd(0));
      output(dma_addr_3)  = high(wrd(0));
      output(dma_addrh_3) = low (wrd(1));
      output(dma_bc_3)    = low (trans_len);
      output(dma_bc_3)    = high(trans_len);
      output(dma_req)     = dma_on_3; /* start DMA channel # 3 */
      end;

      do;             /* channel # 3 , memory to 588(1) */
      output(dma_req)     = dma_off_3;
      output(dma_flff)    = 0;    /* clear first/last flip-flop */
      output(dma_mode)    = dma_tx_mode_3;
      output(dma_addr_3)  = low (wrd(0));
      output(dma_addr_3)  = high(wrd(0));
      output(dma_addrh_3) = low (wrd(1));
      output(dma_bc_3)    = low (trans_len);
      output(dma_bc_3)    = high(trans_len);
      output(dma_req)     = dma_on_3; /* start DMA channel # 3 */
      end;

    end;
  end;
  return;

end dma_load;

/*-------------------------------------------------------------------------*/
```

Figure 7-50. DMA Routine (Continued)

```
/*-------------------------------------------------------------------------*/

dma_load: procedure(channel,direction,trans_len,buff_ptr) reentrant;

/* To load and start the  80186 DMA controller for the desired operation */

  declare dma_rx_mode   literally '1010001001000000b'; /* rx channel */
       /* src=IO, dest=M(inc), sync=src, TC, noint, priority, byte */

  declare dma_tx_mode   literally '0001011010000000b'; /* tx channel */
       /* src=M(inc), dest=IO, sync=dest, TC, noint, noprior, byte */

  declare channel byte;       /* channel # */
  declare direction byte;     /* 0 = rx, 588 -> mem; 1 = tx, mem -> 588 */
  declare trans_len word;     /* byte count */
  declare buff_ptr pointer;   /* buffer pointer in seg:offset form */

  declare  buff_ptr_20bit dword;
  declare ptr1 pointer;
  declare (wrd based ptr1)(2) word;

  ptr1 = @buff_ptr;          /* convert buff_ptr to 20bit buff_ptr */
  buff_ptr_20bit = shl((buff_ptr_20bit := wrd(1)),4) + wrd(0);

  do case channel and 00000001b;
    do case direction and 00000001b;
      do;                   /* channel 0 , 588 to memory */
      outword(dma_0_dpl) = low (buff_ptr_20bit);
      outword(dma_0_dph) = high(buff_ptr_20bit);
      outword(dma_0_spl) = ch_a_588;
      outword(dma_0_sph) = 0;
      outword(dma_0_tc)  = trans_len;
      outword(dma_0_cw)  = dma_rx_mode or 0006h;  /* start DMA channel 0 */
      end;

      do;                   /* channel 0 , memory to 588 */
      outword(dma_0_dpl) = ch_a_588;
      outword(dma_0_dph) = 0;
      outword(dma_0_spl) = low (buff_ptr_20bit);
      outword(dma_0_sph) = high(buff_ptr_20bit);
      outword(dma_0_tc)  = trans_len;
      outword(dma_0_cw)  = dma_tx_mode or 0006h;  /* start DMA channel 0 */
      end;
    end;
```

Figure 7-51. 80186 DMA Routines

```
    do case direction and 00000001b;
      do;                    /* channel 1 , 588 to memory */
      outword(dma_1_dpl) = low (buff_ptr_20bit);
      outword(dma_1_dph) = high(buff_ptr_20bit);
      outword(dma_1_spl) = ch_b_588;
      outword(dma_1_sph) = 0;
      outword(dma_1_tc)  = trans_len;
      outword(dma_1_cw)  = dma_rx_mode or 0006h;  /* start DMA channel 1 */
      end;

      do;                    /* channel 1 , memory to 588 */
      outword(dma_1_dpl) = ch_b_588;
      outword(dma_1_dph) = 0;
      outword(dma_1_spl) = low (buff_ptr_20bit);
      outword(dma_1_sph) = high(buff_ptr_20bit);
      outword(dma_1_tc)  = trans_len;
      outword(dma_1_cw)  = dma_tx_mode or 0006h;  /* start DMA channel 1 */
      end;
    end;
  end;
  return;

end dma_load;

/*-------------------------------------------------------------------------------*/
```

Figure 7-51. 80186 DMA Routines (Continued)

```
/*-----------------------------------------------------------------------*/

intr_588: procedure interrupt 13 reentrant;

  declare event byte;

  enable;
  call read_status_588;
  event = status_588(0) and 00001111b;
  if (status_588(0) and 00100000b) <> 0 /* if execution event */
  then intr_588_flag = 0;               /* reset interrupt flag */

  do case event;

    event_00: ;
    event_01: output(dma_req)  = dma_off_3; /* stop DMA channel # 3 */
    event_02: output(dma_req)  = dma_off_3; /* stop DMA channel # 3 */
    event_03: output(dma_req)  = dma_off_3; /* stop DMA channel # 3 */
    event_04: do;                           /* transmit done */
          output(dma_req)  = dma_off_3;     /* stop DMA channel # 3 */
            trans_count = trans_count + 1;
            if (status_588(2) and 00100000b) <> 0
            then do;
              good_trans_count = good_trans_count + 1;
              coll_cnt(0) = coll_cnt(0) + 1;
              end;
            else do;
              if (status_588(2) and 10000000b) <> 0 /* collision */
              then do;
                  intr_588_flag = 'X';      /* retransmit */
                  coll_cnt(17) = coll_cnt(17) + 1;
                end;
              end;
          end;
    event_05: ;
    event_06: output(dma_req)  = dma_off_3; /* stop DMA channel # 3 */
    event_07: ;
    event_08: do; /* re-initialize rx dma */
            call rcv_disable;

            /* determine receive status */
            rx_buff_off = shl(double(status_588(2)),8)
                + double(status_588(1)) - 2;
            rx_buff_no = low(rx_buff_off / buff_len);
            rx_buff_off = rx_buff_off mod buff_len;
            rx_buff_off = rx_buff_off + 1;
            if rx_buff_off = buff_len   /* status across buff boundaries */
            then do;
              rx_buff_off = 0;
              rx_buff_no = rx_buff_no + 1;
              end;
```

Figure 7-52. Interrupt Service Routine

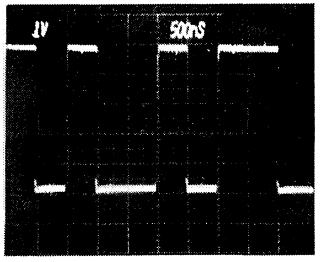
```
            /* update network statistics counters */
            if (buffer_588(rx_buff_no).rx(rx_buff_off) and 00100000b) = 0
            then bad_rcv_count = bad_rcv_count + 1;
            else good_rcv_count = good_rcv_count + 1;

            call rcv_enable(0,@buffer_588(0).rx(0));

          end;
      event_09: call allocate_buffer(new_buffer);
      event_10: output(dma_req)  = dma_off_1; /* stop DMA channel # 1 */
      event_11: ;
      event_12: do; /* re-transmit done */
            output(dma_req)  = dma_off_3; /* stop DMA channel # 3 */
            retrans_count = retrans_count + 1;
            if (status_588(2) and 00100000b) <> 0
            then do;
              good_trans_count = good_trans_count + 1;
              coll_cnt(status_588(1) and 0fh)
              = coll_cnt(status_588(1) and 0fh) + 1;
              end;
            else do;
              if (status_588(2) and 10000000b) <> 0 /* collision */
              then do;
                  intr_588_flag = 'X';  /* retransmit */
                  coll_cnt(17) = coll_cnt(17) + 1;
                end;
              if (status_588(1) and 00100000b) <> 0 /* max coll. */
              then do;
                  intr_588_flag = 0;
                  coll_cnt(16) = coll_cnt(16) + 1;
                end;
              end;
          end;
      event_13: do; /* execution aborted */
            output(dma_req)  = dma_off_3; /* stop DMA channel # 3 */
            intr_588_flag = 'X';
          end;
      event_14: ;
      event_15: ;

    end;

    output (cs_588) = 10000000b;  /* intack, */
    output(pic_ocw2) = seoi_pico; /* specific EOI for 82588 */

    return;

end intr_588;

/*-------------------------------------------------------------------------*/
```

Figure 7-52. Interrupt Service Routine (Continued)

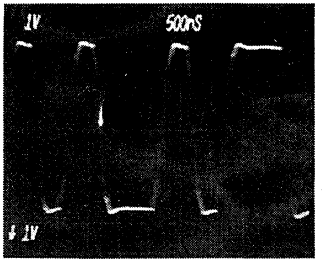
APPENDIX A
StarLAN SIGNALS

Figure 7-53 shows the signals at various points on the StarLAN link as seen on an oscilloscope. The output from the 82588 (1) is a near perfect waveform with square edges. After the driver with slew rate control the rise and fall times increase to about 200 ns (2). After the pulse transformer, on the cable at the driving end the signal is almost sinusoidal (3). At the cable, on the receiving end, the signal is attenuated and slightly distorted (4). However, the zero crossing points are preserved. After passing through a zero crossing receiver the signal is reconstructed back to look like the original waveform (5) generating transitions at the zero crossings. It is important that the receiver must generate edges as close to the zero crossings as possible, otherwise the output of the receiver will have a different high and low time than the original one.

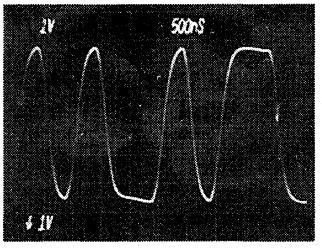
Figure 7-54 shows an eye diagram of the signal at the receiver end for a bit time with the zero-crossing at the center. It could be seen that around the 0.25 and 0.75 microsec. region, where the signal is sampled, 0.6 volts threshold for squelch leaves enough noise margin (of over 0.7 volts).



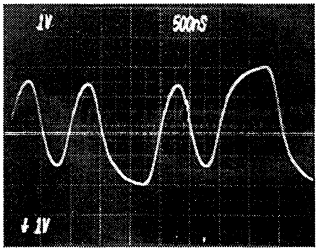
231422-51



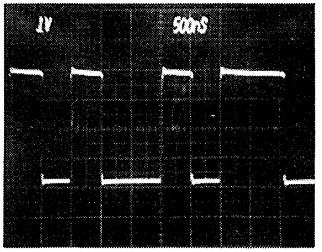
231422-52



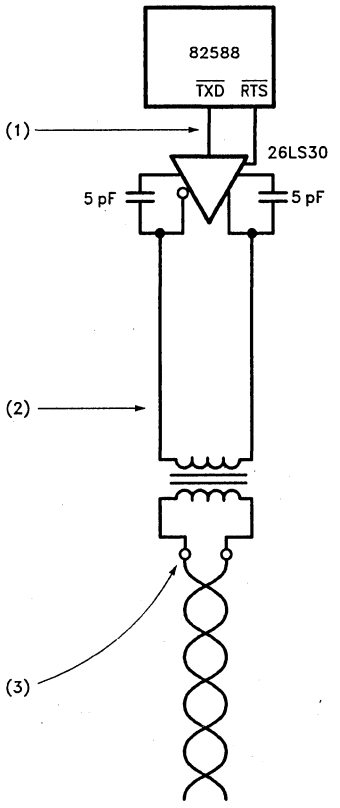
231422-53

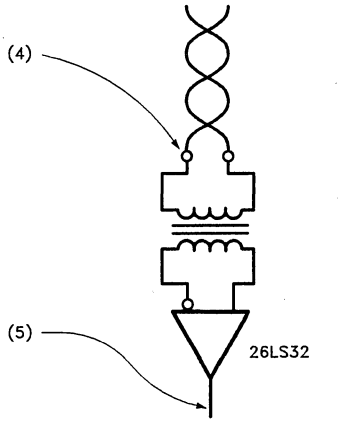


231422-54



231422-55





231422-47

Figure 7-53. StarLAN Signals

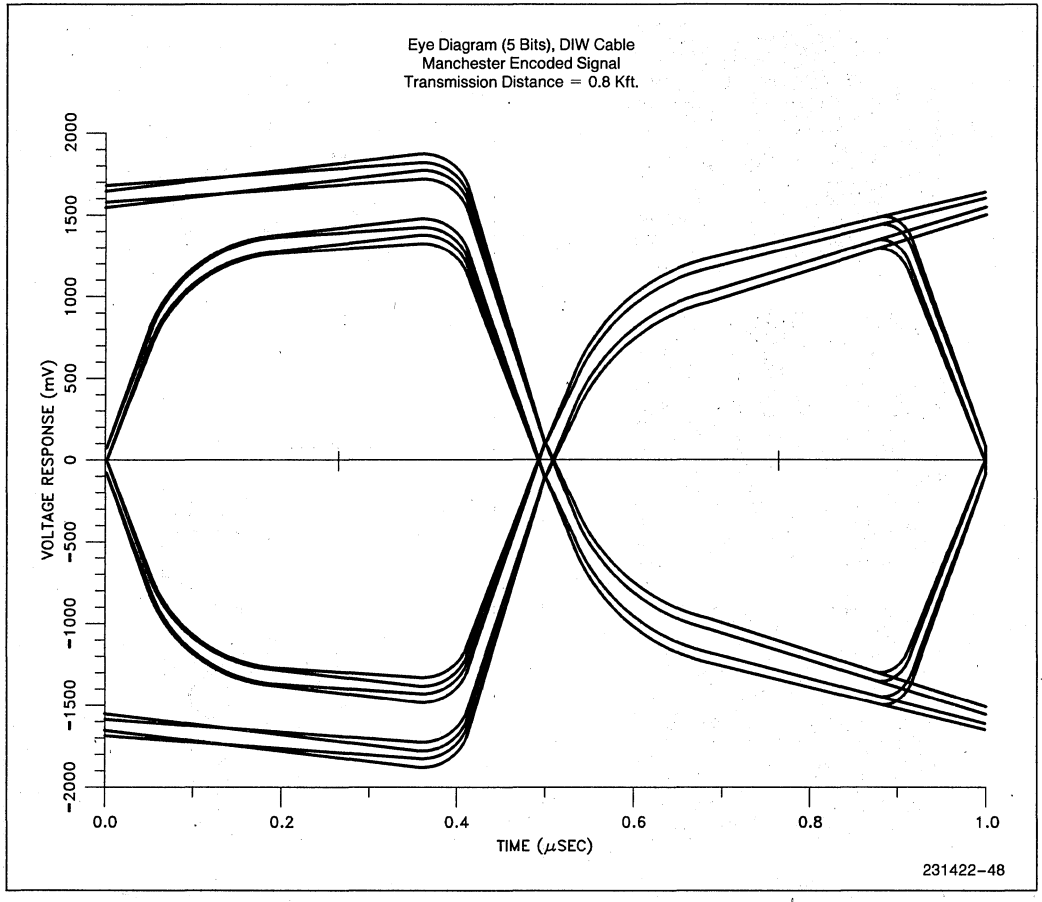


Figure 7-54. Received Signal Eye Diagram

APPENDIX B
SINGLE DMA CHANNEL INTERFACE

In a typical system, the 82588 needs 2 DMA channels to operate in a manner that no received frames are lost as discussed in section 7.5.1.3. If an existing system has only one DMA channel available, it is still possible to operate the 82588 in a way that no frames are lost. This method is recommended only in situations where a second DMA channel is impossible to get.

Figure 7-55 shows how the 82588 DMA logic is interfaced to one channel of a DMA controller. Two DRQ lines are ORed and go to the DMA controller DRQ line and the DACK line from the DMA controller is connected to DACK0 and DACK1 of the 82588. The 82588 is configured for multiple buffer reception (chaining), although the entire frame is received in a single buffer. Let us assume that channel CH-0 is used as the first channel for reception. After the ENAble RECeive command, CH-0 is dedicated to reception. As long as no frame is received, the other channel, CH-1, can be used for executing any commands like transmit, multicast address, dump, etc., by programming the DMA channel for the execution command. The status register should be checked for any ongoing reception, to avoid issuing an execution command when reception is active.

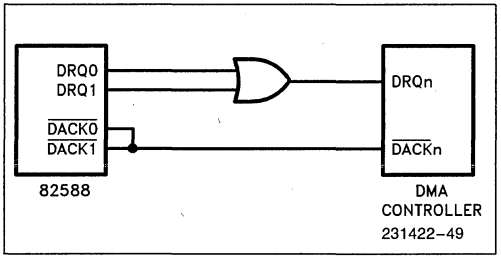


Figure 7-55. 82588 Using One DMA Channel

If a frame is received, an interrupt for additional buffer occurs immediately after an address match is established, as shown in Figure 7-56. After this, the received bytes start filling up the on-chip FIFO. The 82588 activates the DRQ line after 15—FIFO LIMIT + 3 bytes are ready for transfer in the FIFO (about 80 microseconds after the interrupt). The CPU should react to the interrupt within 80 μs and disable the DMA controller. It should also issue an ASSIGN ALTERNATE BUFFER command with INTACK to abort any execution command that may be active. The FIFO fills up in about 160 μs after interrupt. To prevent an underrun, the CPU must reprogram the DMA controller for frame reception and re-enable the DMA controller within 160 μs after the interrupt (time to receive about 21 bytes). No buffer switching actually takes place, although the 82588 generates request for alternate buffer every time in has no additional buffer. The CPU must respond to these interrupts with an ASSIGN ALTERNATE BUFFER command with INTACK. To keep the CPU overhead to a minimum, the buffer size must be configured to the maximum value of 1 kbyte.

If a frame transmission starts deferring due to the reception occurring just prior to an issued transmit command, the transmission can start once the link is free after reception. A maximum of 19 bytes are transmitted (stored in the FIFO and internal registers) followed by a jam pattern and then an execution aborted interrupt occurs. The aborted frame can be transmitted again.

If the transmit command is issued and the 82588 starts transmitting just prior to receiving a frame then transmit wins over receive—but this will obviously lead to a collision.

Note that the interrupt for additional buffer is used to abort an ongoing execution command and to program the DMA channel for reception just when a frame is received. This scheme imposes real time interrupt handling requirements on the CPU and is recommended only when a second DMA channel is not available.

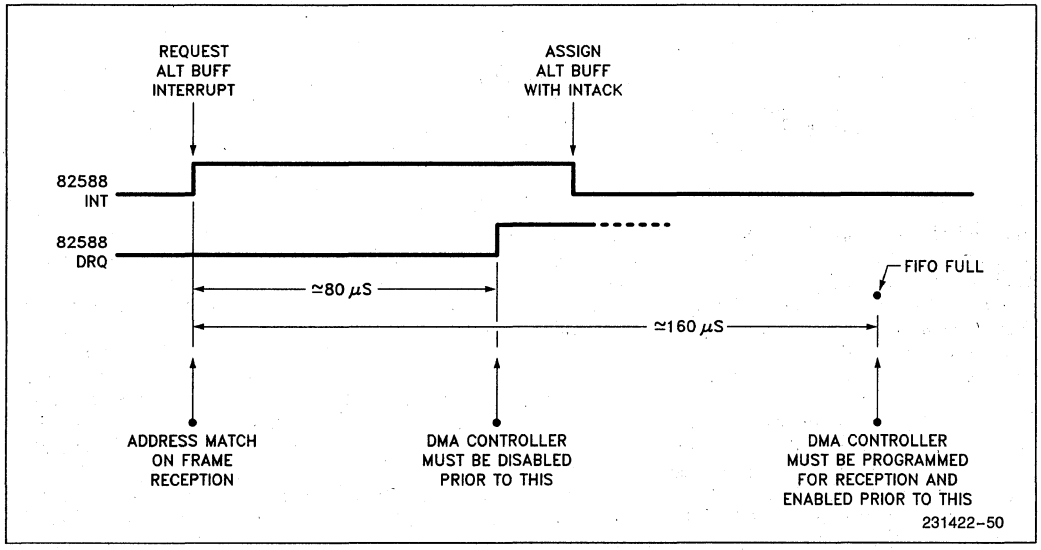


Figure 7-56. Timing at the Beginning of Frame Reception for Single DMA Channel Operation

Local Area Networks Data Sheets

8